Denis Akimov 242044LAFM

Inverse problems for time-fractional wave equation

Master's thesis

Supervisor: Jaan Janno
Professor

Tallinn 2026

Denis Akimov 242044LAFM

Pöördülesanded ajas murrulisele lainevõrrandile

Magistritöö

Juhendaja: Jaan Janno
Professor

Tallinn 2026

Author's declaration of originality and supervisor's resolution

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Denis Akimov
Signature:

19.05.2026

This work corresponds to bachelor's thesis requirements in force.

Supervisor: Jaan Janno
Signature:

19.05.2026

Abstract

Inverse problems for time-fractional wave equation

The goal of this master's thesis was to analyze inverse problems for time-fractional wave equation with Caputo fractional derivatives and attempt to solve them numerically using physics informed neural networks.

Utilizing Laplace transform and Fourier method, analytical solutions were constructed for an equation with non-homogeneous Dirichlet and homogeneous Neumann boundary conditions, as well as an equation with a point source. Uniqueness of parameter estimation for both equations in the complex-frequency domain was established.

Utilizing Gauss-Jacobi quadrature to approximate the Caputo fractional derivative, neural network based solver for the time-fractional wave equation was developed using Python/Tensorflow. With synthetic data generated using numerical Laplace inversion and disturbed with some amount of Gaussian noise, solver showed good estimations and robustness to noise in recovering both the derivative orders and linear parameters when applied to the problem with mixed boundary conditions. On the other hand, the solver failed to converge for an equation with a point source, which shows that without additional generalization PINNs should not be applied to non-smooth problems.

Keywords: inverse problems, fractional derivatives, wave equation, partial differential equations, physics informed neural networks

Annotatsioon

Pöördülesanded ajas murrulisele lainevõrrandile

Käesoleva magistritöö eesmärgiks oli analüüsida ajamuutuja suhtes Caputo murrulisi tuletisi sisaldavale lainevõrrandile püstitatud pöördülesandeid ja lahendada neid numbriliselt füüsikast lähtuvate närvivõrkude abil.

Laplace'i teisenduse ja Fourier meetodi abil koostati analüütilised lahendid võrrandile mittehomogeensete Dirichlet' ja homogeensete Neumanni rajatingimuste korral, ja samuti võrrandile punktallikaga. Mõlema võrrandi puhul näidati parameetrite määramise ühesus komplekssageduste ruumis.

Kasutades Gaussi-Jacobi kvadratuuri Caputo murruliste tuletiste lähendamiseks, koostati ajas murrulise lainevõrrandi jaoks närvivõrkudel põhinev lahendusalgoritm Pythoni ja Tensorflow abil. Numbrilise Laplace'i pöörteisenduse abil genereeritud ja Gaussi müraga häiritud sünteetiliste andmete puhul näitas algoritm häid tulemusi ja mittetundlikkust müra suhtes nii tuletiste järkude kui ka lineaarsete parameetrite määramise ülesanne puhul segarajatingimuste korral. Teisest küljest, algoritm ei koondunud punktallikaga ülesande korral, mis näitab et ilma lisaüldistusteta ei ole närvivõrkude meetod sobilik mittesiledate ülesannete lahendamiseks.

Märksõnad: pöördülesanded, murrulised tuletised, lainevõrrand, osatuletistega diferentsiaalvõrrandid, füüsikast lähtuvad närvivõrgud

List of abbreviations and terms

$\mathbb{N}$	set of natural numbers
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$\mathbb{R}$	set of real numbers
$\mathbb{R}_+$	set of positive real numbers
$\mathbb{C}$	set of complex numbers
C^n	set of functions with n continuous derivatives
L^p	set of functions $f(t)$ such that $\int \lvert f(t)\rvert^p dt < \infty$
A^n	set of functions with absolutely continuous $(n-1)$-th derivative
$\mathcal{L}$	Laplace transform
I_a^α	Riemann-Liouville fractional integral with base a
D_a^α	Riemann-Liouville fractional derivative with base a
${}^C\mathrm{D}^\alpha$	Caputo fractional derivative
$\mathcal{N}$	neuron
$\mathcal{L}^m$	layer with m neurons
$\mathcal{NN}$	neural network
$\odot$	element-wise product of arrays
$\mathbb{1}$	array of ones
PDE	Partial differential equation
PINN	Physics Informed Neural Network
Adam	Gradient descent with adaptive momentum
AdamW	Adam with weight decay regularization

Table of contents

1. Introduction

Fractional calculus is an extension of classical calculus that concerns non-integer orders of repeated differentiation or integration. History of that field can be traced back to 1695, when Leibniz stated that a derivative of order 0.5 "will lead to a paradox, from which one day useful consequences will be drawn" in a letter to L'Hopital [1]. Fractional derivatives in the Caputo sense have found applications in many fields where complex systems that exhibit power-law memory effects emerge. Examples of those applications include modeling properties of linear viscoelastic media in continuum physics [2] and economic growth in finance [3]. Recently there has been a growing interest in applying Caputo fractional derivatives to problems in epidemiology [4] [5] and tumor growth [6].

Most problems that are solved in applied mathematics are so called direct problems, where an effect is inferred from the model and underlying cause. Inverse problems on the other hand ask us to recover the cause and/or the model from observed effect. Such problems are inherently ill-posed: more often than not the solution doesn't exist, cannot be inferred uniquely or shows large sensitivity to noise in the input data [7]. An inverse problem that is often stated in the context of fractional calculus is recovering the non-integer derivative orders, which in the case of time-fractional problems directly characterize weighting of past states in the observed system.

In this thesis, inverse problems for time-fractional wave equations (also known as diffusion-wave equations) were considered. Those equations generalize the classical wave equation by replacing the second time derivative with one or more Caputo derivatives of fractional order between 1 and 2. Laplace transform and Fourier series method were used to construct analytical solutions to postulated equations in the complex-frequency domain and analyze uniqueness of parameter recovery. After that numerical solution to inverse problems was attempted using physics-informed neural networks (PINN), which is a relatively recent and rapidly advancing field in numerical methods and machine learning/artificial intelligence that is applied in solving both forward and inverse problems for partial differential equations [8].

This thesis is structured as follows: Chapter 2 provides a brief overview of fractional calculus in Riemann-Liouville and Caputo sense, as well as physics-informed neural networks and their generalization to fractional models. Inverse problem for time-fractional wave equation with non-homogeneous Dirichlet and homogeneous Neumann boundary conditions was studied in Chapter 3, and time-fractional wave equation with point source was studied in Chapter 4. Link to code repository and some handpicked examples are available in Appendix 2.

2. Background

2.1 Laplace transform

Definition 1. The Laplace transform maps a function $f(t)$ of a real variable t with support on $t > 0$ (time domain) to a function $F(s)$ of a complex variable s (complex-frequency domain). It is defined using the following integral:

$$\mathcal{L}\{f(t)\}(s) = F(s) = \int_0^\infty f(t)e^{-st}\mathrm{d}t, \tag{2.1.1}$$

for those value of s for which the integral makes sense [9].

Properties of the Laplace transform make it a powerful tool for analyzing differential equations, some of those being:

- Derivative in the time domain:

$$\mathcal{L}\left\{\frac{\mathrm{d}^n}{\mathrm{d}t^n}f(t)\right\}(s) = s^nF(s) - \sum_{i=1}^{n} s^{n-i}f^{(i-1)}(0), \tag{2.1.2}$$

  which allows transforming differential equations in time into algebraic equations in the complex frequency domain.
- Derivative in the complex frequency domain:

$$\mathcal{L}\{t^nf(t)\}(s) = (-1)^n\frac{\mathrm{d}^n}{\mathrm{d}s^n}F(s). \tag{2.1.3}$$

- Convolution theorem:

$$\mathcal{L}\left\{\int_0^t f(t-\tau)g(\tau)\mathrm{d}\tau\right\}(s) = F(s)\cdot G(s). \tag{2.1.4}$$

Inversion of the Laplace transform is given by the following integral (known as Bromwich integral or Mellin's inversion formula):

$$\mathcal{L}^{-1}\{F(s)\} = f(t) = \frac{1}{2\pi\mathrm{i}}\int_{\sigma-\mathrm{i}\infty}^{\sigma+\mathrm{i}\infty} e^{st}F(s)\mathrm{d}s, \tag{2.1.5}$$

where real number σ is chosen to be greater than real part of singularities of $F(s)$. Numerical

Laplace transform inversion algorithm as implemented in the Python library "mpmath" applies the linear acceleration method to an alternating series obtained after application of the trapezoidal rule to the Bromwich integral [10]. This implementation was used for data generation in the numerical sections of this thesis.

2.2 Riemann-Lioville fractional calculus

This section contains a brief overview of fractional calculus in Riemann-Liouville and Caputo sense based on [12]. In general fractional calculus studies possibility of generalizing the notion of n-th derivative as n-th power of the derivative operator:

$$\frac{\mathrm{d}^n}{\mathrm{d}x^n} = \frac{\mathrm{d}}{\mathrm{d}x}\frac{\mathrm{d}}{\mathrm{d}x} \dots \frac{\mathrm{d}}{\mathrm{d}x},\ n \in \mathbb{N}, \tag{2.2.1}$$

to rational or real values of n. There are different ways to interpolate in this way, out of which Riemann-Liouville approach utilizes the Gamma function and properties of n-fold integration. Caputo's modification makes Riemann-Liouville's approach more suitable to temporal problems in practical applications.

2.2.1 Gamma function

Definition 2. Gamma function for $z \in \mathbb{C},\ \mathrm{Re}\, z > 0$ can be defined using the following integral:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \mathrm{d}t,$$

and as analytic continuation of the integral function for the rest of the complex plane except zero and negative integers.

With the property $\Gamma(n) = (n-1)!,\ n \in \mathbb{N}$ it is often used to generalize the notion of a factorial to real or complex values of the argument. For this reason it is sometimes called a "continuous factorial". [11]

2.2.2 Riemann-Lioville fractional integral

Definition 3. We define an operator I_a that assigns a function $f(t) \in L^1[a,b]$ to its anti-derivative as:

$$\mathrm{I}_a f(t) = \int_a^t f(\tau)\mathrm{d}\tau,\ a \le t \le b, \tag{2.2.2}$$

and denote n-fold iteration of I_a as I_a^n which would produce an n-th anti-derivative as:

$$\mathrm{I}_a^n f(t) = \int_a^t \int_a^{\tau_1} \int_a^{\tau_2} \ ... \ f(\tau_n)\mathrm{d}\tau_n \ ... \ \mathrm{d}\tau_2\mathrm{d}\tau_1, \ a \leq t \leq b. \tag{2.2.3}$$

Operation of finding n-th anti-derivative of a function can be compressed to only one integration by application of Cauchy's formula for repeated integration. This gives the operator I_a^n a simpler form:

$$\mathrm{I}_a^n f(t) = \frac{1}{(n-1)!}\int_a^t (t-\tau)^{n-1} f(\tau)\mathrm{d}\tau, \ a \leq t \leq b. \tag{2.2.4}$$

In this construction, order of the integration n appears inside the kernel t^{n-1} and the factorial operation $(n-1)!$. If we use the Gamma function to expand the factorial operation to positive real numbers and replace order n with an $\alpha \in \mathbb{R}_+$, we can consider an operator that interpolates between anti-derivatives.

Definition 4. For $\alpha \in \mathbb{R}_+$ an operator I_t^α defined on $f(t) \in L^1[a,b]$ as:

$$\mathrm{I}_a^\alpha f(t) = \frac{1}{\Gamma(\alpha)}\int_a^t (t-\tau)^{\alpha-1} f(\tau)\mathrm{d}\tau, \ a \leq t \leq b, \tag{2.2.5}$$

is called Riemann-Liouville fractional integral of order α. In addition, fractional integral of order $\alpha = 0$ can be defined as being equivalent to the identity map:

$$\mathrm{I}_0^\alpha f(t) = \mathrm{id}_{L^1}. \tag{2.2.6}$$

2.2.3 Riemann-Lioville fractional derivative

Definition 5. We define an operator D^n that assigns a function $f(t) \in C^n$ to its n-th derivative:

$$\mathrm{D}^n f(t) = \frac{\mathrm{d}^n}{\mathrm{d}^n t} f(t). \tag{2.2.7}$$

The relationship between D and I is stated by the Fundamental Theorem of Calculus:

$$\mathrm{D}^1 \mathrm{I}_a^1 f(t) = f(t), \tag{2.2.8}$$

which in turn implies:

$$\mathrm{D}^n \mathrm{I}_a^n f(t) = f(t), \ \forall n \in \mathbb{N}. \tag{2.2.9}$$

If we consider composition of respective operators of order $(m-n)$ and then apply the derivative of order n from both sides, we can find a relationship:

$$\mathrm{D}^n f(t) = \mathrm{D}^m \mathrm{I}_a^{m-n} f(t), \tag{2.2.10}$$

which essentially means that we can recover an n-th derivative by differentiating $m-n$-th anti-derivative m times. This suggests a possibility of obtaining an $\alpha \in \mathbb{R}_+$ order derivative by applying an integer order derivative to Riemann-Liouville fractional integral.

Definition 6. Let $\alpha \in \mathbb{R}_+$ and $n = \lceil \alpha \rceil$. Then an operator D_a^α defined on $f(t) \in L^1[a,b]$ such that:

$$\mathrm{D}_a^\alpha f(t) = \frac{1}{\Gamma(n-\alpha)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \int_a^t (t-\tau)^{n-\alpha-1} f(\tau) \mathrm{d}\tau \tag{2.2.11}$$

is called Riemann-Liouville fractional derivative of order α.

Let us set base point $a = 0$ and denote $\mathrm{D}_0^\alpha = \mathrm{D}^\alpha$. There are a few curious caveats when it comes to applying Riemann-Liouville to time-domain problems. To begin with, fractional derivative of a power function t^k can be computed analytically as:

$$\mathrm{D}^\alpha t^k = \frac{\Gamma(k+1)}{\Gamma(k-\alpha+1)} t^{k-\alpha}, \tag{2.2.12}$$

which is a straightforward generalization of power function derivative rule. We can evaluate this when $k = 0$, which results in:

$$\mathrm{D}^\alpha t^0 = \mathrm{D}^\alpha 1 = \frac{1}{\Gamma(1-\alpha)} t^{-\alpha}. \tag{2.2.13}$$

It appears that Riemann-Liouville fractional derivative of a constant is never zero, so when applied to model time-domain problems the derivative terms will not disappear even when the observed system is in a steady-state.

In addition if an $n = \lceil \alpha \rceil$ order time-domain differential equation is constructed with Riemann-Liouville fractional derivatives, the existence and uniqueness theorem demands supplying n initial conditions in the form:

$$\begin{cases} \mathrm{D}^{\alpha-k} f(0) = b_k, \ k = 1, 2, 3, ..., n-1, \\ \mathrm{I}_0^{n-\alpha} f(0) = b_n. \end{cases} \tag{2.2.14}$$

It would be very difficult to give interpretation of those values and pre-supply them in most practical applications. Those problems motivate further modification of the Riemann-Liouville definition.

2.2.4 Caputo fractional derivative

Definition 7. By $A^n[a, b]$ we denote a set of functions $f(t)$ with an absolutely continuous $(n-1)$-st derivative, which means that there exists an integrable function $f^{(n)}(t) \in L^1[a, b]$ such that:

$$f^{(n-1)}(t) = f^{(n-1)}(a) + \int_a^t f^{(n)}(\tau) \mathrm{d}\tau. \tag{2.2.15}$$

Definition 8. Let $\alpha \in \mathbb{R}_+$ and $n = \lceil \alpha \rceil$. An operator ${}^C\mathrm{D}^\alpha$ defined on $f(t) \in A^n[0, \infty]$ by swapping the order of operators in definition 6 such that:

$${}^C\mathrm{D}^\alpha f(t) = \mathrm{I}_0^{n-\alpha} \mathrm{D}^n f(t) = \frac{1}{\Gamma(n-\alpha)} \int_0^t (t-\tau)^{n-\alpha-1} f^{(n)}(\tau) \mathrm{d}\tau, \tag{2.2.16}$$

is called the Caputo fractional derivative.

Immediate consequence of the modified definition is that Caputo derivative of a constant function is unconditionally zero. Using the fact that the Caputo operator is essentially a convolution integral, we can use the convolution theorem (2.1.4) to find the Laplace transform representation. Considering:

$$\mathcal{L}\{t^{n-\alpha-1}\}(s) = \frac{\Gamma(n-\alpha)}{s^{n-\alpha}}, \tag{2.2.17}$$

and the time-domain derivative property of the Laplace transform (2.1.2) we can deduce:

$$\mathcal{L}\{{}^{C}\mathrm{D}^{\alpha}f(t)\}(s) = s^{\alpha}F(s) - \sum_{i=1}^{n} s^{\alpha-i}f^{(i-1)}(0). \tag{2.2.18}$$

It can be seen that the initial conditions $f^{(i-1)}(0),\ i = 1, 2, ..., \lceil\alpha\rceil$ are the same that a classical differential equation of order $\lceil\alpha\rceil$ would require supplying. This removes any initial condition related difficulty from generalizing dynamical systems to fractional cases.

2.2.5 Time-fractional wave equation

General form of the non-local wave equation can be given as follows:

$$\int_0^t \eta(t-\tau)u''(\tau)\mathrm{d}\tau - ku_{xx} = f(x,t), \tag{2.2.19}$$

where $k \in \mathbb{R}_+$ and kernel $\eta(t)$ represents weighting of past states in the system. For $1 < \alpha < 2$ we choose a scaled power law as the weighting kernel:

$$\eta(t) = \frac{t^{1-\alpha}}{\Gamma(2-\alpha)}, \tag{2.2.20}$$

which represents a long-range memory effect. As a result we obtain a wave equation with a Caputo fractional derivative:

$${}^{C}\mathrm{D}^{\alpha}u - ku_{xx} = f(x,t). \tag{2.2.21}$$

Equations of this form are called time-fractional wave equations or fractional diffusion-wave equations. Last name is motivated by the interpolation property of the fractional derivative. For a suitable function $f(t) \in A^2[0,\infty]$:

$$\lim_{\alpha\to 2^+} {}^{C}\mathrm{D}^{\alpha}f(t) = f''(t), \tag{2.2.22}$$

$$\lim_{\alpha\to 1^-} {}^{C}\mathrm{D}^{\alpha}f(t) = f'(t) - f'(0), \tag{2.2.23}$$

and in case $1 < \alpha < 2$ the Caputo fractional derivative intepolates between hyperbolic wave equation and parabolic diffusion equation.

Another choice of the weighting functions with $1 < \alpha_2 < \alpha_1 < 2$ and $a \in \mathbb{R}_+$:

$$\eta(t) = \frac{t^{1-\alpha_1}}{\Gamma(2-\alpha_1)} + a\frac{t^{1-\alpha_2}}{\Gamma(2-\alpha_2)}, \tag{2.2.24}$$

where $a \in \mathbb{R}_+$. As a result we obtain a wave equation with two Caputo fractional derivatives as:

$$^{C}\mathrm{D}^{\alpha_1}u + a\,{}^{C}\mathrm{D}^{\alpha_2}u - ku_{xx} = f(x,t). \tag{2.2.25}$$

In continuum physics equations of type (2.2.21) emerge in the analysis of impact waves in viscoelastic materials that exhibit a power-law creep [2]. It can also be used for modeling oscillations of a cable made of smart materials, where the second term in (2.2.25) may represent resistance of the medium the cable is emerged in [13]. Further in this thesis specific examples of an equation (2.2.25) will be considered.

2.3 Physics Informed Neural Networks

This section provides a brief overview of PINNs, which is a rapidly developing tool for solving both forward and inverse problems for ordinary or partial differential equations utilizing recent advancements in the fields of machine learning and artificial neural networks. Recent studies show that while finite element methods usually perform better and/or faster for low dimensional problems, PINNs scale much better into the higher dimensions [14] [15]. Flexibility of the framework also allows fast implementation of inverse problems with minimal coding changes. The tool set remains very accessible thanks to development of open source machine learning libraries like Tensorflow [16], PyTorch [17] and JAX [18]. Out of these choices, Tensorflow was used for code in this thesis due to previous experience with the library.

2.3.1 Automatic differentiation

Automatic differentiation (also known as automatic gradient or autograd) is an algorithm for computing derivatives with machine precision. A directed computational graph of a differentiable function $f : \mathbb{R}^m \to \mathbb{R}$ acting on an input variable $\mathbf{x} \in \mathbb{R}^m$ is constructed based on the assumption that it can be decomposed into a finite sequence of intermediate variables v_i and differentiable computation steps f_i so that $v_i = f_i(\{v_j : \ j \neq i\})$. During evaluation of the function $f(\mathbf{x})$ in what is usually called a forward pass, values of all intermediate variables in the graph are recorded.

Process called back-propagation moves backwards through the graph and assigns to every variable v_i a derivative:

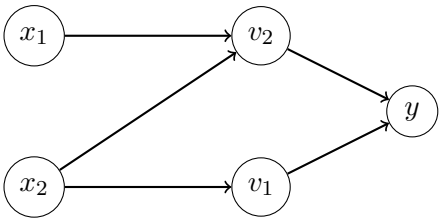

Figure 2.1. Example of a simple computational graph for some function of two variables $f(x_1, x_2) = y$

$$\hat{v}_i = \frac{\partial y}{\partial v_i}, \tag{2.3.1}$$

representing sensitivity of the output y to the variable v_i. Using the Figure 2.1 as an example, back-propagation would start at y and compute:

$$\hat{v}_1 = \frac{\partial y}{\partial v_1}, \tag{2.3.2}$$

$$\hat{v}_2 = \frac{\partial y}{\partial v_2}, \tag{2.3.3}$$

and then using the chain rule evaluate:

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial v_1}\frac{\partial v_1}{\partial x_1} = \hat{v}_1\frac{\partial v_1}{\partial x_1}, \tag{2.3.4}$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial v_1}\frac{\partial v_1}{\partial x_2} + \frac{\partial y}{\partial v_2}\frac{\partial v_2}{\partial x_2} = \hat{v}_1\frac{\partial v_1}{\partial x_2} + \hat{v}_2\frac{\partial v_2}{\partial x_2}, \tag{2.3.5}$$

which would be the output of the automatic differentiation algorithm [19].

Tensorflow library handles automatic differentiation through a gradient tape environment, which will record the forward pass of any computation result made inside the environment and then on request back-propagate to differentiate the result with respect to some observed variable. Higher order derivatives can be computed by using nested gradient tapes [20] (example 1 in Appendix 2).

2.3.2 Neural networks

Definition 9. Neuron is a map $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}$ defined as:

$$\mathcal{N}(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b), \tag{2.3.6}$$

where $\mathbf{w}$ is a vector of weights, b is bias and f is the activation function.

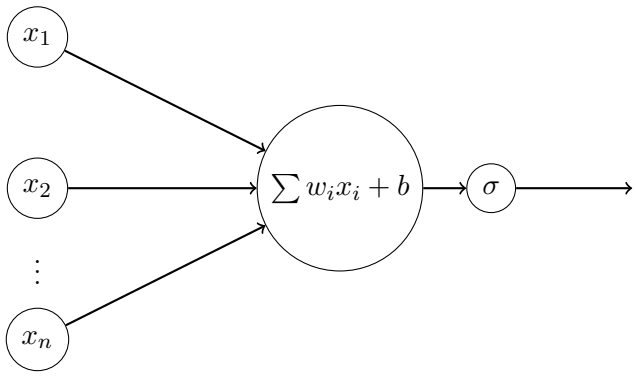

Figure 2.2. Visual representation of a single artificial neuron

Common choices of activations include [21]:

- Identity function:

$$\mathrm{id}(x) = x. \tag{2.3.7}$$

- Logistic sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.3.8}$$

- ReLU (Rectified Linear Unit):

$$\mathrm{ReLU}(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}. \tag{2.3.9}$$

- Hyperbolic tangent:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3.10}$$

Definition 10. Layer is a map $\mathcal{L}^m : \mathbb{R}^n \to \mathbb{R}^m$ defined using a sequence of neurons $\{\mathcal{N}_i\}_{i=1}^m$ with respective weights $\mathbf{w}_i$ and biases b_i so that:

$$\mathcal{L}^m(\mathbf{x})_i = \mathcal{N}_i(\mathbf{x}),\ i = 1,\ 2,\ ...,\ m-1,\ m. \tag{2.3.11}$$

Number of neurons in the layer m is called "width" of the layer.

Definition 11. Given an input layer $\mathcal{L}_{in}^{n_1} : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_1}$, an output layer $\mathcal{L}_{out}^{n_{out}} : \mathbb{R}^{n_{L+1}} \to \mathbb{R}^{n_{out}}$ and a sequence of L hidden layers $\{\mathcal{L}_i^{n_{i+1}} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}\}_{i=1}^{L}$, multi-layer neural network is a map $\mathcal{NN} : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}$ defined using composition of layers:

$$\mathcal{NN} = \mathcal{L}_{out}^{n_{out}} \circ \mathcal{L}_L^{n_{L+1}} \circ \mathcal{L}_{L-1}^{n_L} \circ \ ... \ \circ \mathcal{L}_1^{n_2} \circ \mathcal{L}_{in}^{n_1}. \tag{2.3.12}$$

Number of hidden layers in the network plus output layer $D = L + 1$ is often called "depth" of the network, and maximum number of neurons per layer W is often called "width" of the network. Networks with $D = 2$ are referred to as "shallow" and networks with $D > 2$ are referred to as "deep". Capabilities of neural networks to approximate arbitrary continuous functions that occur in different practical applications are described by the so called Universal Approximation Theorems [22].

Different modifications and possible improvements on neural network architectures are designed for each specific task. An architecture proposed in [23] for physics informed machine learning includes two additional transformer layers $\mathcal{V}^m$ and $\mathcal{U}^m$. Given input vector $\mathbf{x}$, output of a hidden layer $\mathbf{y}_i = \mathcal{L}_i^m(\hat{\mathbf{y}}_{i-1})$ is modified as:

$$\hat{\mathbf{y}}_i = (\mathbb{1} - \mathbf{y}_i) \odot \mathcal{U}^m(\mathbf{x}) + \mathbf{y}_i \odot \mathcal{V}^m(\mathbf{x}). \tag{2.3.13}$$

The construction is designed to explicitly account for multiplicative interactions as well introduce residual connections. Residual connections were first introduced to improve learning of very deep neural networks used for image recognition tasks, which were prone to degradation of accuracy [24]. Implementation code for this architecture can be seen on Example 2 in Appendix 2.

2.3.3 Loss function

Training a neural network $\mathcal{NN}_{\mathbf{W}}(\mathbf{x})$, where $\mathbf{W}$ represents all weights and biases in the network, is essentially an optimization problem. Given a target functional $\mathbb{L}(\mathbf{W})$ known as the loss function, we want to find $\mathbf{W}^*$ so that:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathbb{L}(\mathbf{W}). \tag{2.3.14}$$

We can consider a general PDE of the form:

$$\begin{cases} \mathrm{A}u(\mathbf{x}) = f(\mathbf{x}), \ \mathbf{x} \in \Omega, \\ \mathrm{B}u(\mathbf{x}) = g(\mathbf{x}), \ \mathbf{x} \in \partial\Omega, \end{cases} \tag{2.3.15}$$

where Ω is the domain and $\partial\Omega$ is the boundary of Ω. Operators A and B are some differential operators that can be, in most cases, computed directly from the network using automatic differentiation. We define a neural network $\tilde{u}(\mathbf{x}) = \mathcal{NN}(\mathbf{x})$ as a smooth approximation of the correct solution $u(\mathbf{x})$. As a potential optimization target we can evaluate mean squared residuals:

$$\mathbb{L}_r = \frac{1}{\mu(\Omega)} \int_\Omega (\mathrm{A}\tilde{u}(\mathbf{x}) - f(\mathbf{x}))^2 \mathrm{d}\mathbf{x}, \tag{2.3.16}$$

$$\mathbb{L}_b = \frac{1}{S(\partial\Omega)} \int_{\partial\Omega} (\mathrm{B}\tilde{u}(\mathbf{x}) - g(\mathbf{x}))^2 \mathrm{d}S. \tag{2.3.17}$$

We can uniformly sample N_Ω collocation points on the domain Ω and $N_{\partial\Omega}$ points on the boundary. Integrals (2.3.16) and (2.3.17) could then be approximated by applying Monte-Carlo integration [25]:

$$\mathbb{L}_r \approx \frac{1}{N_\Omega} \sum_{i=1}^{N_\Omega} (\mathrm{A}\tilde{u}(\mathbf{x}_i) - f(\mathbf{x}_i))^2, \tag{2.3.18}$$

$$\mathbb{L}_b \approx \frac{1}{N_{\partial\Omega}} \sum_{i=1}^{N_{\partial\Omega}} (\mathrm{B}\tilde{u}(\mathbf{x}_i) - g(\mathbf{x}_i))^2. \tag{2.3.19}$$

If some measurement data $u_{data}(\mathbf{x}^*)$ is available for N_{data} points $\mathbf{x}^* \in \Omega$ we can introduce the data loss as mean squared error:

$$\mathbb{L}_d = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (\tilde{u}(\mathbf{x}_i^*) - u_{data}(\mathbf{x}_i^*))^2, \tag{2.3.20}$$

Finally, the loss function for training PINNs is constructed as a linear combination of several desirable targets:

$$\mathbb{L} = \lambda_1 \mathbb{L}_r + \lambda_2 \mathbb{L}_b + \lambda_3 \mathbb{L}_d. \tag{2.3.21}$$

It can be seen that (2.3.21) represents a multi-objective optimization problem. In order to avoid converging in a local minimum where only one loss term is satisfied, careful tuning of scaling coefficients λ_i is required [26]. Apart from manual tuning, several adaptive methods have been proposed. One class of methods evaluates balancing based on current or past loss statistics and records a moving average of coefficients with smoothing β at every training step t:

$$\hat{\lambda}_i(t) = F_i(\mathbb{L}_0(t), \mathbb{L}_1(t)...), \tag{2.3.22}$$

$$\lambda_i(t) = \beta\lambda_i(t-1) + (1-\beta)\hat{\lambda}_i(t). \tag{2.3.23}$$

Examples of such methods include learning rate annealing [23], dynamic norm-based balancing [27] and Relative Loss Balancing with Random Lookback (ReLoBRaLo) [26].

Some methods treat coefficients λ_i as learnable variables (which means they will be changed by the optimization algorithm alongside network weights $\mathbf{W}$) and control their behavior by adding penalty terms to the total loss function:

$$\mathbb{L}^* = \mathbb{L} + \sum_{i=1}^{m} F_i(\lambda_i). \tag{2.3.24}$$

Examples of those methods include uncertainty based multi-task learning with upper bounds [28] and GradNorm [29].

After the loss function is constructed with the suitable tuning algorithm, it is to be optimized with respect to weights and biases $\mathbf{W}$. Widely used optimizer in many applications is gradient descent [25], which updates $\mathbf{W}$ at every training step t using rule:

$$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta\nabla_\gamma\mathbb{L}(\mathbf{W^t}). \tag{2.3.25}$$

In the context of machine learning training step t is often called an "epoch" and step size η is often called "learning rate". Nowadays most popular choices of an optimizer include modifications of gradient descent like Adam [31] and Adam-W [32] which use moving averages of gradients (dubbed "momentum") for their update rules. Fine tuning of models is usually done using second-order optimizers like L-BFGS [33].

2.3.4 Inverse problems with PINNs

Assuming that a forward problem solver is constructed, and that sufficient measurement data is available for the loss term (2.3.20), it is not difficult to readjust the framework for inverse problems. With total loss function defined as in (2.3.21), we consider an optimization problem for network weights $\mathbf{W}$ and parameters $\boldsymbol{\gamma}$ that are to be inferred: find $\mathbf{W}^*$ and $\boldsymbol{\gamma}^*$ so that:

$$\begin{cases} \mathbf{W}^* = \arg\min_{\mathbf{W}} \mathrm{L}(\mathbf{W}, \boldsymbol{\gamma}) \\ \boldsymbol{\gamma}^* = \arg\min_{\boldsymbol{\gamma}} \mathrm{L}(\mathbf{W}, \boldsymbol{\gamma}) \end{cases}. \tag{2.3.26}$$

In this sense parameters $\boldsymbol{\gamma}$ are considered learnable, and are to be updated at every training epoch with either the same or a different optimizer as network weights $\mathbf{W}$. For implementation of one training step see Example 3 in Appendix 2.

Optimizer algorithms as implemented in Tensorflow are tuned for unconstrained optimization. In order to enforce constraints on the inferred parameters, raw trainable variables are to be projected into the area of interest using a smooth and bijective map. Examples of such maps include:

- Sigmoid

$$p = a + (b - a) \cdot \sigma(p_{raw}),\ p_{raw} \in \mathbb{R},\ p \in (a, b), \tag{2.3.27}$$

- Softplus

$$p = \ln(1 + e^{p_{raw}}),\ p_{raw} \in \mathbb{R},\ p \in \mathbb{R}_+ \tag{2.3.28}$$

- Exponential

$$p = e^{p_{raw}},\ p_{raw} \in \mathbb{R},\ p \in \mathbb{R}_+. \tag{2.3.29}$$

From different alternatives for enforcing positivity exponential function can result in superior convergence in some cases [30].

2.3.5 PINNs with fractional derivatives

With all that in mind, we are interested in applying PINNs to solving inverse problems for PDEs involving an operator:

$$ {}^{C}\mathrm{D}^{\alpha} f(t) = \frac{1}{\Gamma(2-\alpha)} \int_0^t (t-\tau)^{\alpha-1} f''(\tau) \mathrm{d}\tau, \ 1 < \alpha < 2. \tag{2.3.30} $$

Since automatic differentiation cannot be used here (chain rule is generalized to arbitrary order α by Faa di Bruno's formula [12], but it is quite cumbersome to be used for back-propagation), more conventional methods have to be applied. Application of Gauss-Jacobi quadrature is proposed in [34], in which Caputo fractional derivative or order $1 < \alpha < 2$ is approximated using:

$$ \begin{aligned} {}^{C}\mathrm{D}^{\alpha} f(t) \approx \frac{1}{\Gamma(2-\alpha)} \Bigg[& \frac{f'(t) - f'(0)}{t^{\alpha-1}} - (\alpha-1) \frac{f(t) - f(0) - t f'(t)}{t^{\alpha}} \\ & + \alpha(\alpha-1) t^{2-\alpha} \left(\sum_{i=1}^{n} w_i \frac{f(t) - f(t - t\tau_i) - t\tau_i f'(t)}{(t\tau_i^2)} \right) \Bigg], \end{aligned} \tag{2.3.31} $$

where w_i and τ_i are the quadrature weights and nodes that depend on n and α. Fast convergence of Gauss-Jacobi quadrature allows to reduce the computational and memory costs of evaluating Caputo derivatives.

The weights and nodes can be computed using Jacobi weights function from the SciPy library [35]. Thus during the forward pass the neural network is evaluated at N_{qd} past points defined with $(x, t - t\tau_i)$ for every collocation point (x, t), and the Caputo derivative of the neural network at (x, t) is computed using rule (2.3.31). At the same time, solving an inverse problem for α requires back-propagation through the quadrature, but computational graph cannot be constructed as SciPy and Tensorflow speak different languages. In fact this will result in an exception being thrown in the code. Knowing sensitivity of some computational step $v(w, \tau)$ to changes in weights w and nodes τ we need to manually construct:

$$ \frac{\partial v}{\partial \alpha} = \frac{\partial v}{\partial w} \frac{\partial w}{\partial \alpha} + \frac{\partial v}{\partial \tau} \frac{\partial \tau}{\partial \alpha}. \tag{2.3.32} $$

One option is to set:

$$ \frac{\partial w}{\partial \alpha} = 0, \ \frac{\partial \tau}{\partial \alpha} = 0, \tag{2.3.33} $$

which would effectively erase one term in the derivative of (2.3.31) with respect to α. If sensitivity to changes in nodes and weights is to be back-propagated after all, we can evaluate central differences:

$$\frac{\partial w}{\partial \alpha} = \frac{w(\alpha + h) - w(\alpha - h)}{2h}, \tag{2.3.34}$$

$$\frac{\partial \tau}{\partial \alpha} = \frac{\tau(\alpha + h) - \tau(\alpha - h)}{2h}, \tag{2.3.35}$$

for some small h. This manual computation can then be inserted into the computational graph using Tensorflow's custom gradient wrapper.

A possible alternative to using quadrature methods is solving the inverse problem in the Laplace transform domain. This was utilized for time-fractional diffusion equation (Caputo derivative of order $0 < \alpha < 1$) in [36], where Laplace transform inversion algorithm on the real line was used to move the network output into the time domain and compare with input data. Solving the wave equation with derivatives of order $1 < \alpha < 2$ would require constructing a neural network $\mathcal{NN} : \mathbb{R} \times \mathbb{C} \to \mathbb{C}$ that can approximate $U(x, s)$ for $s \in \mathbb{C}$, especially along the contours used by the chosen Laplace inversion algorithm. Complex-valued neural networks is a field that has recently been successful in various areas [37], but they were not considered in this thesis.

3. Inverse problem for time-fractional wave equation with mixed boundary conditions

In this chapter we will consider the following time-fractional wave equation with inhomogeneous Dirichlet boundary condition at $x = 0$ and homogeneous Neumann boundary condition at $x = l$:

$$\begin{cases} {}^{C}\mathrm{D}^{\alpha_1}u(x,t) + a\,{}^{C}\mathrm{D}^{\alpha_2}u(x,t) - ku_{xx}(x,t) = 0,\ x \in (0,l),\ t > 0; \\ u(0,t) = g(t),\ u_x(l,t) = 0,\ t > 0, \\ u(x,0) = u_t(x,0) = 0,\ x \in (0,l). \end{cases} \tag{3.0.1}$$

We assume that $1 < \alpha_2 < \alpha_1 < 2$. We also assume that $a, k \in \mathbb{R}_+$ and $g(t) \in A^2[0,\infty]$.

3.1 Analysis of the inverse problem

3.1.1 Eigenvalues and eigenfunctions of the Laplace operator for mixed BC

In order to apply Fourier series method to the problem we must first find eigenvalues and eigenfunctions of the operator $-\frac{\partial^2}{\partial x^2}$ subject to mixed boundary conditions, or in other words find non-trivial solutions to:

$$\begin{cases} \phi_{xx} = -\lambda\phi \\ \phi(0) = \phi'(l) = 0 \end{cases} \tag{3.1.1}$$

It is trivial that only $\phi(x) = 0$ solves (3.1.1) in case $\lambda = 0$. In case $\lambda < 0$ general solution to that equation has the form:

$$\phi(x) = Ae^{\sqrt{-\lambda}x} + Be^{-\sqrt{-\lambda}x}. \tag{3.1.2}$$

At $x = 0$ Dirichlet boundary condition enforces:

$$\phi(0) = A + B = 0, \tag{3.1.3}$$

at the same time at $x = l$ Neumann boundary condition enforces:

$$\phi_x(l) = C_1\sqrt{-\lambda}e^{\sqrt{-\lambda}x} - C_2\sqrt{-\lambda}e^{-\sqrt{-\lambda}x}. \tag{3.1.4}$$

Putting them together results in:

$$A\sqrt{-\lambda}(e^{\sqrt{-\lambda}x} + e^{-\sqrt{-\lambda}x}) = 0 \implies A = 0 \implies B = 0, \tag{3.1.5}$$

and for negative eigenvalues λ we have only a trivial eigenfunction $\phi(x) = 0$. In case $\lambda > 0$ general solution to the equation takes the form:

$$\phi(x) = A\cos(\sqrt{\lambda}x) + B\sin(\sqrt{\lambda}x). \tag{3.1.6}$$

Dirichlet boundary condition at $x = 0$ enforces:

$$\phi(0) = A + 0 = 0 \implies \phi(x) = B\sin(\sqrt{\lambda}x) \tag{3.1.7}$$

Now we consider the Neumann boundary condition at $x = l$:

$$\phi_x(l) = B\sqrt{\lambda}\cos(\sqrt{\lambda}l) = 0, \tag{3.1.8}$$

$$\cos(\sqrt{\lambda}l) = 0, \tag{3.1.9}$$

$$\sqrt{\lambda}l = \frac{(2n+1)\pi}{2},\ n \in \mathbb{N}_0. \tag{3.1.10}$$

With this equation (3.1.1) produces eigenvalues and eigenfunctions in the form:

$$\lambda_n = \frac{(2n+1)^2\pi^2}{4l^2}, \tag{3.1.11}$$

$$\phi_n(x) = B\sin\left(\frac{(2n+1)\pi x}{2l}\right). \tag{3.1.12}$$

3.1.2 Forward problem

To deal with a non-homogeneous boundary condition in 3.0.1 we make a substitution:

$$u = v + g(t), \tag{3.1.13}$$

where the new function $v(x,t)$ satisfies the homogeneous conditions. Plugging it into (3.0.1) results in an equation for v:

$$\begin{cases} {}^{C}\mathrm{D}^{\alpha_1} v(x,t) + a\, {}^{C}\mathrm{D}^{\alpha_2} v(x,t) - k v_{xx}(x,t) = -[{}^{C}\mathrm{D}^{\alpha_1} g(t) + aDc^{\alpha_2} g(t)],\ x \in (0,l),\ t > 0, \\ v(0,t) = v_x(l,t) = 0,\ t > 0, \\ v(x,0) = -g(0),\ v_t(x,0) = -g'(0). \end{cases} \tag{3.1.14}$$

Using (3.1.12) and (3.1.11) we can expand $v(x,t)$ and $v_{xx}(x,t)$ into the Fourier sine series:

$$v(x,t) = \sum_{n=0}^{\infty} v_n(t) \sin\left(\frac{(2n+1)\pi x}{2l}\right), \tag{3.1.15}$$

$$v_{xx}(x,t) = \sum_{n=0}^{\infty} \lambda_n v_n(t) \sin\left(\frac{(2n+1)\pi x}{2l}\right), \tag{3.1.16}$$

where the coefficients $v_n(t)$ can be computed as:

$$v_n(t) = \frac{2}{l} \int_0^l v(x,t) \sin\left(\frac{(2n+1)\pi x}{2l}\right) \mathrm{d}x. \tag{3.1.17}$$

To proceed with the Fourier method we must also expand initial conditions and the forcing term, which are constant in x. Let us consider series expansion of a function $f(x) = C$ for $x \in (0,l)$. We define an odd reflection:

$$\tilde{f}(x) = \begin{cases} C,\ x \in (0,l), \\ -C,\ x \in (-l,0), \end{cases} \tag{3.1.18}$$

and consider the sine series coefficients:

$$f_n = \frac{1}{l}\int_{-l}^{l} \tilde{f}(x)\sin\left(\frac{(2n+1)\pi x}{2l}\right)\mathrm{d}x = \frac{2C}{l}\int_{0}^{l}\sin\left(\frac{(2n+1)\pi x}{2l}\right)\mathrm{d}x = \frac{4C}{(2n+1)\pi}. \quad (3.1.19)$$

This results in a sine series expansion of a constant function $f(x) = C$ as:

$$f(x) = \frac{4}{\pi}\sum_{n=0}^{\infty}\frac{C}{(2n+1)}\sin\left(\frac{(2n+1)\pi x}{2l}\right). \quad (3.1.20)$$

Let us denote the operator $[\mathrm{D}_t^{\alpha_1} + a\mathrm{D}_t^{\alpha_2}]$ as L. Expanding all terms in the equation 3.1.14 into the Fourier sine series results in an equation:

$$\sum_{n=0}^{\infty}\left[\mathrm{L}v_n(t) + \frac{4}{(2n+1)\pi}\mathrm{L}g(t) + k\lambda_n v_n(t) + \right]\sin\left(\frac{(2n+1)\pi x}{2l}\right) = 0, \quad (3.1.21)$$

$$\mathrm{L}v_n(t) + \frac{4}{(2n+1)\pi}\mathrm{L}g(t) + k\lambda_n v_n(t) = 0, \quad (3.1.22)$$

with initial conditions:

$$v_n(0) = -\frac{4g(0)}{(2n+1)\pi},\ v_n'(0) = -\frac{4g'(0)}{(2n+1)\pi}. \quad (3.1.23)$$

Laplace transform of the operator L would be:

$$\mathscr{L}[\mathrm{L}f] = F(s)\left[s^{\alpha_1} + as^{\alpha_2}\right] - f(0)\left[s^{\alpha_1-1} + as^{\alpha_2-1}\right] - f'(0)\left[s^{\alpha_1-2} + as^{\alpha_2-2}\right], \quad (3.1.24)$$

and applying it to a function $f = v_n(t) + \frac{4g(t)}{(2n+1)\pi}$ results in:

$$\begin{aligned}\mathscr{L}[\mathrm{L}f] = &\left(V_n(s) + \frac{4G(s)}{(2n+1)\pi}\right)\left[s^{\alpha_1} + as^{\alpha_2}\right] \\ &- \left(-\frac{4g(0)}{(2n+1)\pi} + \frac{4g(0)}{(2n+1)\pi}\right)\left[s^{\alpha_1-1} + as^{\alpha_2-1}\right] \\ &- \left(-\frac{4g'(0)}{(2n+1)\pi} + \frac{4g'(0)}{(2n+1)\pi}\right)\left[s^{\alpha_1-2} + as^{\alpha_2-2}\right].\end{aligned} \quad (3.1.25)$$

After cancellation of several terms we are left with:

$$\mathscr{L}[\mathrm{L}f] = \left(V_n(s) + \frac{4G(s)}{(2n+1)\pi}\right)\left[s^{\alpha_1} + as^{\alpha_2}\right]. \tag{3.1.26}$$

Thus applying Laplace transform to the equation (3.1.22) results in the following algebraic equation:

$$\left(V_n(s) + \frac{4G(s)}{(2n+1)\pi}\right)\left[s^{\alpha_1} + as^{\alpha_2}\right] + k\frac{(2n+1)^2\pi^2}{4l^2}V_n(s) = 0, \tag{3.1.27}$$

and solving for $V_n(s)$ gives us:

$$V_n(s) = -\frac{4G(s)}{\pi}\frac{\left[s^{\alpha_1} + as^{\alpha_2}\right]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n}\frac{1}{(2n+1)}, \tag{3.1.28}$$

By applying Laplace transform directly to the substitution (3.1.13) we obtain:

$$U(x,s) = V(x,s) + G(s), \tag{3.1.29}$$

and thus the solution to equation (3.0.1) in the complex frequency domain takes the form:

$$U(x,s) = G(s)\left[1 - \frac{4}{\pi}\sum_{n=0}^{\infty}\frac{\left[s^{\alpha_1} + as^{\alpha_2}\right]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n}\frac{1}{(2n+1)}\sin\left(\frac{(2n+1)\pi x}{2l}\right)\right]. \tag{3.1.30}$$

Time domain solution $u(x,t)$ can be obtained by application of the Bromwich integral to $U(x,s)$.

3.1.3 Determining α_2

For this inverse problem we know $g(t)$ and the solution at the opposite edge $u(l,t) = q(t)$. We want to obtain variables k and a, as well as fractional derivative orders α_1 and α_2. We assume, that $q(t)$ is well-behaved and has a Laplace transform:

$$Q(s) = \int_0^{\infty} q(t)e^{-st}\mathrm{d}t \tag{3.1.31}$$

The solution (3.1.30) at $x = l$ takes the form:

$$\frac{\pi}{4}\left(1 - \frac{Q(s)}{G(s)}\right) = \sum_{n=0}^{\infty} \frac{[s^{\alpha_1} + as^{\alpha_2}]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)} \tag{3.1.32}$$

Let us denote $\frac{\pi}{4}\left(1 - \frac{Q(s)}{G(s)}\right)$ as $T(s)$ and factor out s^{α_2}:

$$T(s) = s^{\alpha_2} \sum_{n=0}^{\infty} \frac{[s^{\alpha_1 - \alpha_2} + a]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)}, \tag{3.1.33}$$

We can find an expression for α_2 using absolute value and natural logarithm:

$$|T(s)| = |s|^{\alpha_2} \left| \sum_{n=0}^{\infty} \frac{[s^{\alpha_1 - \alpha_2} + a]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)} \right|, \tag{3.1.34}$$

$$\ln|T(s)| = \alpha_2 \ln s + \ln \left| \sum_{n=0}^{\infty} \frac{[s^{\alpha_1 - \alpha_2} + a]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)} \right|, \tag{3.1.35}$$

$$\alpha_2 = \frac{\ln|T(s)|}{\ln s} - \frac{\ln \left| \sum_{n=0}^{\infty} \frac{[s^{\alpha_1 - \alpha_2} + a]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)} \right|}{\ln(s)}. \tag{3.1.36}$$

At this point we are interested in behavior of the numerator of the second term in above expression. Since $\alpha_2 < \alpha_1$ by assumption, in the limit process $s \to 0$ the series under the logarithm approaches:

$$\sum_{n=0}^{\infty} \frac{a}{k\lambda_n} \frac{(-1)^n}{(2n+1)} = \sum_{n=0}^{\infty} \frac{a}{k} \frac{4l^2}{(2n+1)^2\pi^2} \frac{(-1)^n}{(2n+1)} = \sum_{n=0}^{\infty} \frac{4al^2}{k\pi^2} \frac{(-1)^n}{(2n+1)^3}. \tag{3.1.37}$$

By comparison test:

$$\sum_{n=0}^{\infty} \left| \frac{(-1)^n}{(2n+1)^3} \right| = \sum_{n=0}^{\infty} \frac{1}{(2n+1)^3} \leq \sum_{n=1}^{\infty} \frac{1}{n^3} < \infty \tag{3.1.38}$$

the series converges absolutely and thus the numerator stays bounded as $s \to 0$. With this in mind:

$$\alpha_2 = \lim_{s\to 0} \frac{\ln|T(s)|}{2\ln(s)}, \tag{3.1.39}$$

and α_2 is uniquely defined. We multiply (3.1.33) by $s^{-\alpha_2}$ and evaluate the limit process $s \to 0$ again:

$$s^{-\alpha_2}T(s) = \sum_{n=0}^{\infty} \frac{[s^{\alpha_1-\alpha_2} + a]}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)}, \tag{3.1.40}$$

$$\lim_{s\to 0} s^{-\alpha_2}T(s) = \sum_{n=0}^{\infty} \frac{4al^2}{k\pi^2} \frac{(-1)^n}{(2n+1)^3}, \tag{3.1.41}$$

from which it follows that ratio $\frac{a}{k}$ is uniquely defined.

3.1.4 Determining a and k

Returning to the expression of $T(s)$:

$$T(s) = \sum_{n=0}^{\infty} \frac{s^{\alpha_1} + as^{\alpha_2}}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)}, \tag{3.1.42}$$

we can consider its value at $s = 1$:

$$T(1) = \sum_{n=0}^{\infty} \frac{1+a}{1+a+k\lambda_n} \frac{(-1)^n}{(2n+1)}. \tag{3.1.43}$$

Let us denote $k\lambda_n = \tau a(2n+1)^2$, where parameter τ is defined as:

$$\tau = \frac{k}{a}\frac{\pi^2}{4l^2}. \tag{3.1.44}$$

We define a function $\xi(\epsilon)$ that characterizes sensitivity of $T(1)$ to the value of a:

$$\xi(\epsilon) = \sum_{n=0}^{\infty} \frac{1+\epsilon}{1+\epsilon+\tau x(2n+1)^2} \frac{(-1)^n}{(2n+1)}. \tag{3.1.45}$$

For a to be uniquely defined from $T(1)$, function $\xi(\epsilon)$ has to be bijective for $\epsilon > 0$. Let us compute the derivative of ξ:

$$\begin{aligned}
\xi'(\epsilon) &= \sum_{n=0}^{\infty} \frac{(1+\epsilon+\epsilon\tau(2n+1)^2) - (1+\epsilon)(1+\tau(2n+1)^2)}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2} \frac{(-1)^n}{(2n+1)} \\
&= \sum_{n=0}^{\infty} \frac{\epsilon\tau(2n+1)^2 - (1+\epsilon)\tau(2n+1)^2}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2} \frac{(-1)^n}{(2n+1)} \\
&= \sum_{n=0}^{\infty} \frac{-\tau(2n+1)^2}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2} \frac{(-1)^n}{(2n+1)} \\
&= -\tau \sum_{n=0}^{\infty} \frac{(2n+1)(-1)^n}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2}.
\end{aligned} \tag{3.1.46}$$

In the case $\epsilon > 0$ the series is absolutely convergent and $\tau > 0$ insures all singularities are left in $\epsilon \leq 0$. To ensure bijectivity of $\xi(\epsilon)$ we can consider the sign of the derivative. Using the Laplace transform we can rewrite:

$$\frac{1}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2} = \int_0^{\infty} te^{-t(1+\epsilon+\tau\epsilon(2n+1)^2)} \mathrm{d}t, \tag{3.1.47}$$

$$\sum_{n=0}^{\infty} \frac{(2n+1)(-1)^n}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2} = \sum_{n=0}^{\infty} (2n+1)(-1)^n \int_0^{\infty} te^{-t(1+\epsilon+\tau\epsilon(2n+1)^2)} \mathrm{d}t, \tag{3.1.48}$$

$$\sum_{n=0}^{\infty} \frac{(2n+1)(-1)^n}{(1+\epsilon+\tau\epsilon(2n+1)^2)^2} = \sum_{n=0}^{\infty} \int_0^{\infty} (2n+1)(-1)^n te^{-t(1+\epsilon+\tau\epsilon(2n+1)^2)} \mathrm{d}t. \tag{3.1.49}$$

We can evaluate absolute convergence:

$$\sum_{n=0}^{\infty} \int_0^{\infty} \left|(2n+1)te^{-t(1+\epsilon+\tau\epsilon(2n+1)^2)}\right| \mathrm{d}t \leq \sum_{n=0}^{\infty} \int_0^{\infty} nte^{-tn^2} \mathrm{d}t = \sum_{n=0}^{\infty} \frac{1}{n^3} < \infty, \tag{3.1.50}$$

which allows us to swap the sum and the integral:

$$\sum_{n=0}^{\infty} \int_0^{\infty} (2n+1)(-1)^n te^{-t(1+\epsilon+\tau\epsilon(2n+1)^2)} dt = \int_0^{\infty} te^{-t(1+\epsilon)} \sum_{n=0}^{\infty} (2n+1)(-1)^n e^{-t\tau\epsilon(2n+1)^2} \mathrm{d}t. \tag{3.1.51}$$

Now we study the series:

$$\sum_{n=0}^{\infty}(2n+1)(-1)^n e^{-t\tau\epsilon(2n+1)^2}. \tag{3.1.52}$$

This form is related to derivative of the Jacobi Theta function $\theta_2(z,q)$ [38]:

$$\theta_2(z,q) = 2\sum_{n=0}^{\infty} q^{(n+1/2)^2}\cos[(2n+1)z] = 2\sum_{n=0}^{\infty} q^{\frac{(2n+1)^2}{4}}\cos[(2n+1)z], \tag{3.1.53}$$

$$\frac{\mathrm{d}}{\mathrm{d}z}\theta_2(z,q) = -2\sum_{n=0}^{\infty}(2n+1)q^{\frac{(2n+1)^2}{4}}\sin[(2n+1)z]. \tag{3.1.54}$$

Evaluating $\theta_2(z,q)$ at $z=\frac{\pi}{2}$ and $q=e^{-4t\tau\epsilon}$ results in:

$$\theta_2'\left(\frac{\pi}{2}, e^{-4t\tau\epsilon}\right) = -2\sum_{n=0}^{\infty}(2n+1)(-1)^n e^{-t\tau\epsilon(2n+1)^2}, \tag{3.1.55}$$

$$\sum_{n=0}^{\infty}(2n+1)(-1)^n e^{-t\tau\epsilon(2n+1)^2} = -\frac{1}{2}\theta_2'\left(\frac{\pi}{2}, e^{-4t\tau\epsilon}\right). \tag{3.1.56}$$

On the other hand $\theta_2(z,q)$ has a product form:

$$\theta_2(z,q) = 2q^{\frac{1}{4}}\cos(z)\prod_{n=1}^{\infty}(1-q^{2n})[1+2q^{2n}\cos(2z)+q^{4n}] \tag{3.1.57}$$

Evaluating the derivative with respect to z produces:

$$\frac{\mathrm{d}}{\mathrm{d}z}\theta_2(z,q) = -2q^{\frac{1}{4}}\sin(z)\prod_{n=1}^{\infty}(1-q^{2n})[1-2q^{2n}\cos(2z)+q^{4n}] + 2q^{\frac{1}{4}}\cos(z)\cdot r, \tag{3.1.58}$$

$$r = \left(\prod_{n=1}^{\infty}(1-q^{2n})[1+2q^{2n}\cos(2z)+q^{4n}]\right)'. \tag{3.1.59}$$

If we evaluate this expression at $z = \frac{\pi}{2}$, second term disappears as $\cos(\frac{\pi}{2}) = 0$ and we are left with:

$$\begin{aligned}\theta_2'\left(\frac{\pi}{2}, q\right) &= -2q^{\frac{1}{4}} \prod_{n=1}^{\infty}(1-q^{2n})[1-2q^{2n}+q^{4n}] \\ &= -2q^{\frac{1}{4}} \prod_{n=1}^{\infty}(1-q^{2n})(1-q^{2n})^2 \\ &= -2q^{\frac{1}{4}} \prod_{n=1}^{\infty}(1-q^{2n})^3.\end{aligned} \tag{3.1.60}$$

As a result we can write the series (3.1.52) as a product:

$$\sum_{n=0}^{\infty}(2n+1)(-1)^n e^{-t\tau\epsilon(2n+1)^2} = e^{-t\tau\epsilon} \prod_{n=1}^{\infty}(1-e^{-8t\tau\epsilon n})^3. \tag{3.1.61}$$

With this the derivative $\xi'(\epsilon)$ can be written down as:

$$\xi'(\epsilon) = -\tau \int_0^{\infty} te^{-t(1+\epsilon-t\tau\epsilon)} \prod_{n=1}^{\infty}(1-e^{-8t\tau\epsilon n})^3 \mathrm{d}t. \tag{3.1.62}$$

Since for $t, \epsilon, \tau > 0$ all functions under the integral are positively defined, and thus $\xi'(\epsilon)$ stays negative for $\epsilon > 0$. This means that $\xi(\epsilon)$ is monotonically decreasing function that can be inverted. With that in mind we can recover:

$$a = \xi^{-1}(T(1)), \tag{3.1.63}$$

and a is uniquely defined. Since a and ratio $\frac{a}{k}$ are uniquely defined, k is also uniquely defined.

3.1.5 Determining α_1

Once again we return to $T(s)$ and compute its derivative with respect to s:

$$T(s) = \sum_{n=0}^{\infty} \frac{s^{\alpha_1} + as^{\alpha_2}}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n} \frac{(-1)^n}{(2n+1)}. \tag{3.1.64}$$

$$
\begin{aligned}
T'(s) = &\sum_{n=0}^{\infty} \frac{(\alpha_1 s^{\alpha_1 - 1} + \alpha_2 a s^{\alpha_2 - 1})(s^{\alpha_1} + a s^{\alpha_2} + k\lambda_n)}{(s^{\alpha_1} + a s^{\alpha_2} + k\lambda_n)^2} \frac{(-1)^n}{(2n+1)} \\
&- \sum_{n=0}^{\infty} \frac{(s^{\alpha_1} + a s^{\alpha_2})(\alpha_1 s^{\alpha_1 - 1} + \alpha_2 a s^{\alpha_2 - 1})}{(s^{\alpha_1} + a s^{\alpha_2} + k\lambda_n)^2} \frac{(-1)^n}{(2n+1)}
\end{aligned}
\tag{3.1.65}
$$

Evaluating at $s = 1$:

$$
\begin{aligned}
T'(1) &= \sum_{n=0}^{\infty} \frac{(\alpha_1 + \alpha_2 a)(1 + a + k\lambda_n) - (1+a)(\alpha_1 + \alpha_2 a)}{(1 + a + k\lambda_n)^2} \frac{(-1)^n}{(2n+1)} \\
&= \sum_{n=0}^{\infty} \frac{(\alpha_1 + \alpha_2 a) k\lambda_n}{(1 + a + k\lambda_n)^2} \frac{(-1)^n}{(2n+1)}.
\end{aligned}
\tag{3.1.66}
$$

As a result:

$$
\frac{T'(1)}{\alpha_1 + \alpha_2 a} = \sum_{n=0}^{\infty} \frac{k\lambda_n}{(1 + a + k\lambda_n)^2} \frac{(-1)^n}{(2n+1)},
\tag{3.1.67}
$$

and since α_2, a, and c are defined uniquely it follows that α_1 is also uniquely defined.

3.2 Numerical experiments

In this section we consider a specific example of the equation (3.0.1):

$$
\begin{cases}
{}^{C}\mathrm{D}^{1.8} u(x,t) + {}^{C}\mathrm{D}^{1.3} u(x,t) - 4u_{xx}(x,t) = 0, \ x \in (0,3), \ t > 0, \\
u(0,t) = [\cos(6t) - \cos(2t)] e^{-0.3t}, \ u_x(3,t) = 0, \ t > 0, \\
u(x,0) = u_t(x,0) = 0, \ x \in (0,3).
\end{cases}
\tag{3.2.1}
$$

Ground truth values for inferred parameters α_1, α_2, a and k are respectively 1.8, 1.3, 1.0 and 4.0. Raw parameters α_1 and α_2 are projected into $(1, 2)$ using logistic sigmoid, raw parameters a and k are projected into $\mathbb{R}_+$ using exponential.

3.2.1 Methodology

Reference data $u^*(t)$ at $x = 3$ was generated by applying numerical inverse Laplace transform to the analytical solution (3.1.30) and then disturbing it with some amount of Gaussian noise.

Solution $\tilde{u}(x,t) = \mathcal{NN}(x,t)$ was approximated using an improved neural network architecture defined in (2.3.13) with depth $D = 3$ and width $W = 64$.

Loss function was constructed as follows:

$$\mathbb{L} = \lambda_0 \mathbb{L}_r + \lambda_1 \mathbb{L}_{ic1} + \lambda_2 \mathbb{L}_{ic2} + \lambda_3 \mathbb{L}_{bc1} + \lambda_4 \mathbb{L}_{bc2} + \lambda_5 \mathbb{L}_{data}, \tag{3.2.2}$$

where the individual objectives were:

$$\mathbb{L}_r = \frac{1}{N_\Omega} \sum_{i=1}^{N_\Omega} \frac{1}{\sqrt{\epsilon + t_i}} ({}^C\mathrm{D}^{\alpha_1} \tilde{u}(x_i,t_i) + a\, {}^C\mathrm{D}^{\alpha_2} u(x_i,t_i) - k u_{xx}(x_i,t_i))^2, \tag{3.2.3}$$

$$\mathbb{L}_{ic1} = \frac{1}{N_{\Omega_0}} \sum_{i=1}^{N_{\Omega_0}} (\tilde{u}(x,0))^2, \tag{3.2.4}$$

$$\mathbb{L}_{ic2} = \frac{1}{N_{\Omega_0}} \sum_{i=1}^{N_{\Omega_0}} (\tilde{u}_t(x,0))^2, \tag{3.2.5}$$

$$\mathbb{L}_{bc1} = \frac{1}{N_{\partial\Omega}} \sum_{i=1}^{N_{\partial\Omega}} (\tilde{u}(0,t) - [\cos(6t) - \cos(2t)] e^{-0.3t})^2, \tag{3.2.6}$$

$$\mathbb{L}_{bc2} = \frac{1}{N_{\partial\Omega}} \sum_{i=1}^{N_{\Omega_0}} (\tilde{u}_x(3,t))^2, \tag{3.2.7}$$

$$\mathbb{L}_{data} = \frac{1}{N_{Data}} \sum_{i=1}^{N_{Data}} (\tilde{u}(3,t) - u^*(t))^2. \tag{3.2.8}$$

In this loss function, derivatives u_t, u_x and u_{xx} were computed using automatic differentiation and Caputo derivatives were computed using Gauss-Jacobi quadrature with 50 nodes. With that $N_\Omega = 2500$ collocation points were sampled in the domain $(0,3) \times (0,10)$, $N_{\partial\Omega} = 500$ time points for the boundaries $x = 0$ and $x = 3$, and $N_{\Omega_0} = 500$ spatial points for initial conditions $t = 0$. Points were sampled using Latin hypercube uniform sampling.

Relative Loss Balancing with Random Lookback proposed in [26] was used for tuning the loss term coefficients λ_i. Balancing based on loss progress between learning steps t and t' is defined using:

$$\lambda_i^{bal}(t,t') = m \cdot \frac{\exp\left(\frac{\mathbb{L}_i t}{\mathcal{T}\mathbb{L}_i t'}\right)}{\sum_{j=0}^{m}\exp\left(\frac{\mathbb{L}_j t}{\mathcal{T}\mathbb{L}_j t'}\right)},\ i = 0,\ 1,\ \ldots,\ m-1,\ m \tag{3.2.9}$$

where $\mathcal{T}$ is a hyperparameter. A Bernoulli random variable ρ with $\mathrm{E}(\rho) \approx 1$ is rolled (representing the occasional lookback at the initial loss values), and the coefficients are then updated using:

$$\hat{\lambda}_i(t) = \rho\lambda_i(t-1) + (1-\rho)\lambda_i^{bal}(t,0), \tag{3.2.10}$$

$$\lambda_i(t) = \beta\hat{\lambda}_i(t) + (1-\beta)\lambda_i^{bal}(t,t-1). \tag{3.2.11}$$

Model was trained for 25000 epochs using Adam optimizer with learning rate $\eta_{\mathbf{w}} = 10^{-3}$. Paramaters were trained using Adam-W optimiser (Adam with L2 regularization) with default learning rate of $\eta_{pars} = 10^{-3}$. The first 1000 epochs corresponded to a "warm up" stage when η_{pars} was temporarily set to 10^{-8}. This prevented the optimizer from trying to compensate initialization loss by drifting the parameters. All hyperparameters and selected values can be seen on Table 1.

Epochs	25000
Warmup	1000
D	3
W	64
N_Ω	2500
$N_{\partial\Omega}$	1000
N_{Ω_0}	1000
$\eta_{\mathbf{w}}$	10^{-3}
η_{pars}	10^{-3}
$E(\rho)$	0.999
β	0.99
$\mathcal{T}$	10^{-3}

Table 1. All training hyperparameters and chosen values

3.2.2 Results with 1% noise

In this experiment 5 independent training runs were made, and each time the synthetic data was disturbed with 1% Gaussian noise. Shape of the input during the last run can be seen on Figure 3.1. Figure 3.2 shows median of the loss components.

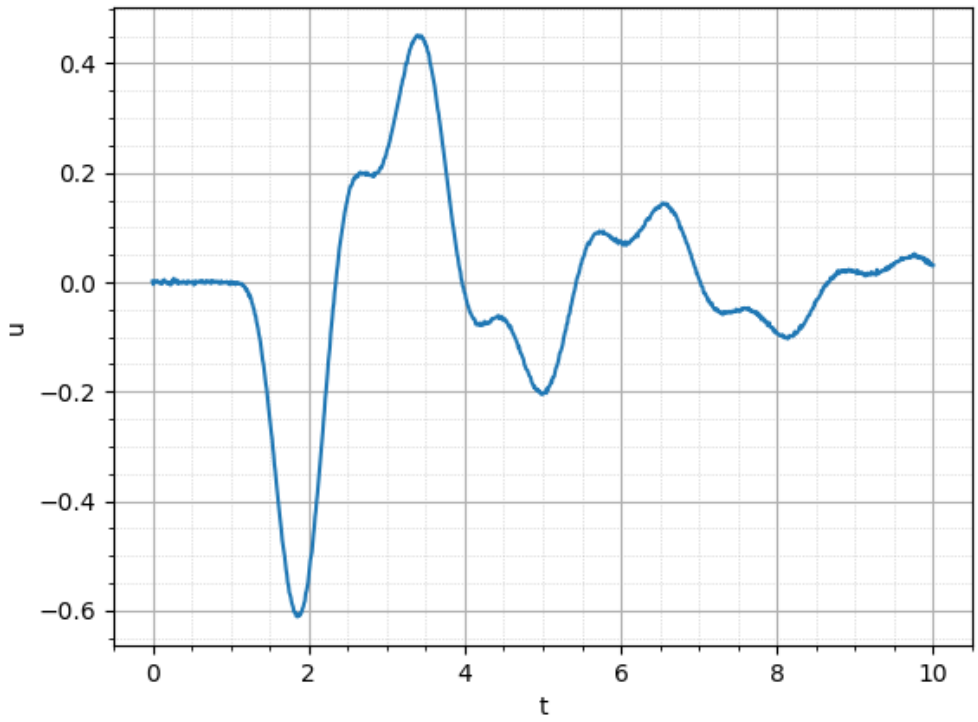

Figure 3.1. Synthetic data with 1% Gaussian noise

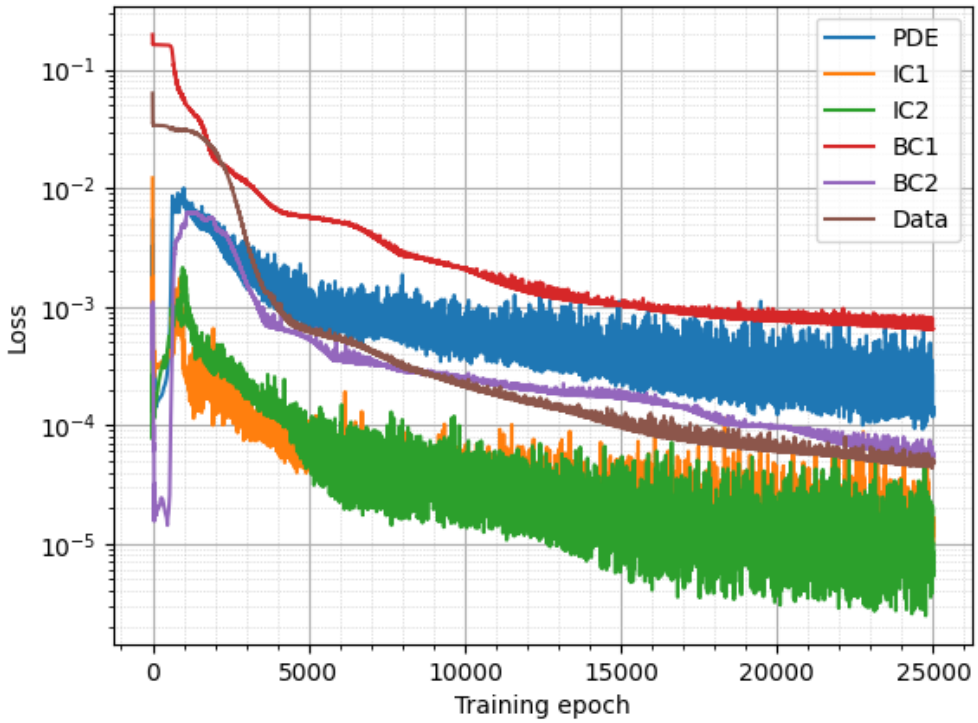

Figure 3.2. Median loss components over 5 independent training runs

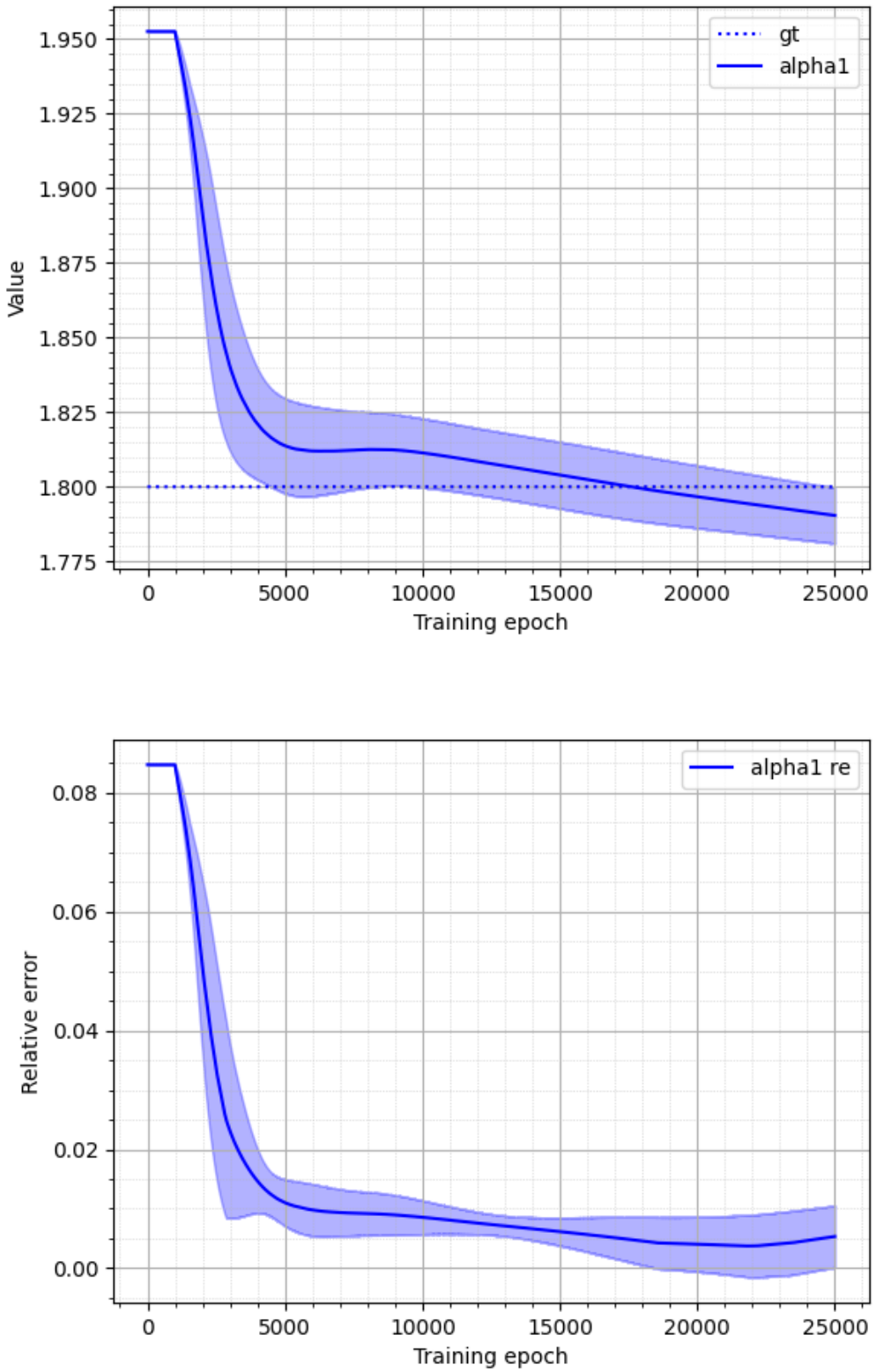

Figure 3.3. Mean and standard deviation of parameter α_1 over 5 independent training runs

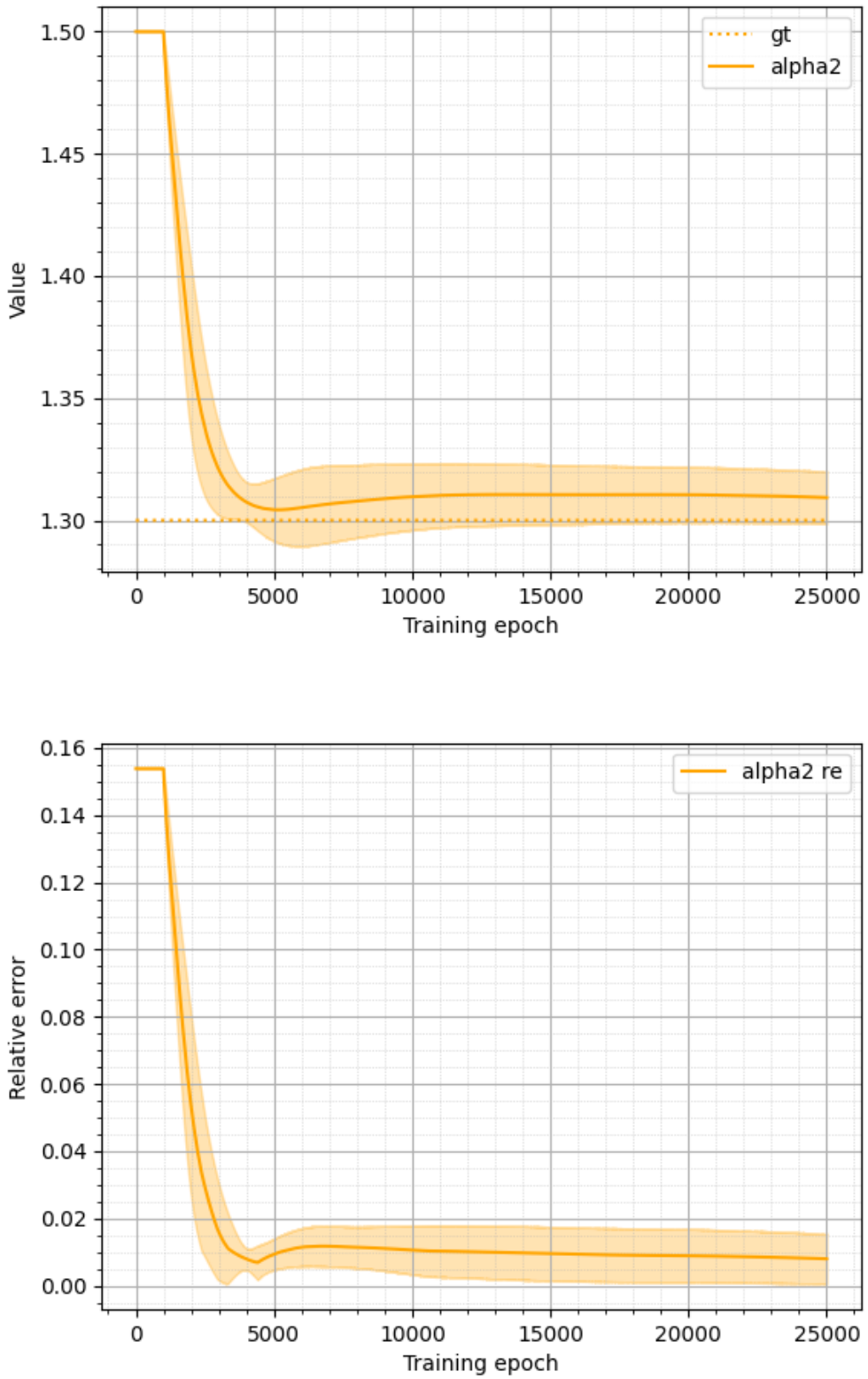

Figure 3.4. Mean and standard deviation of parameter α_2 over 5 independent training runs

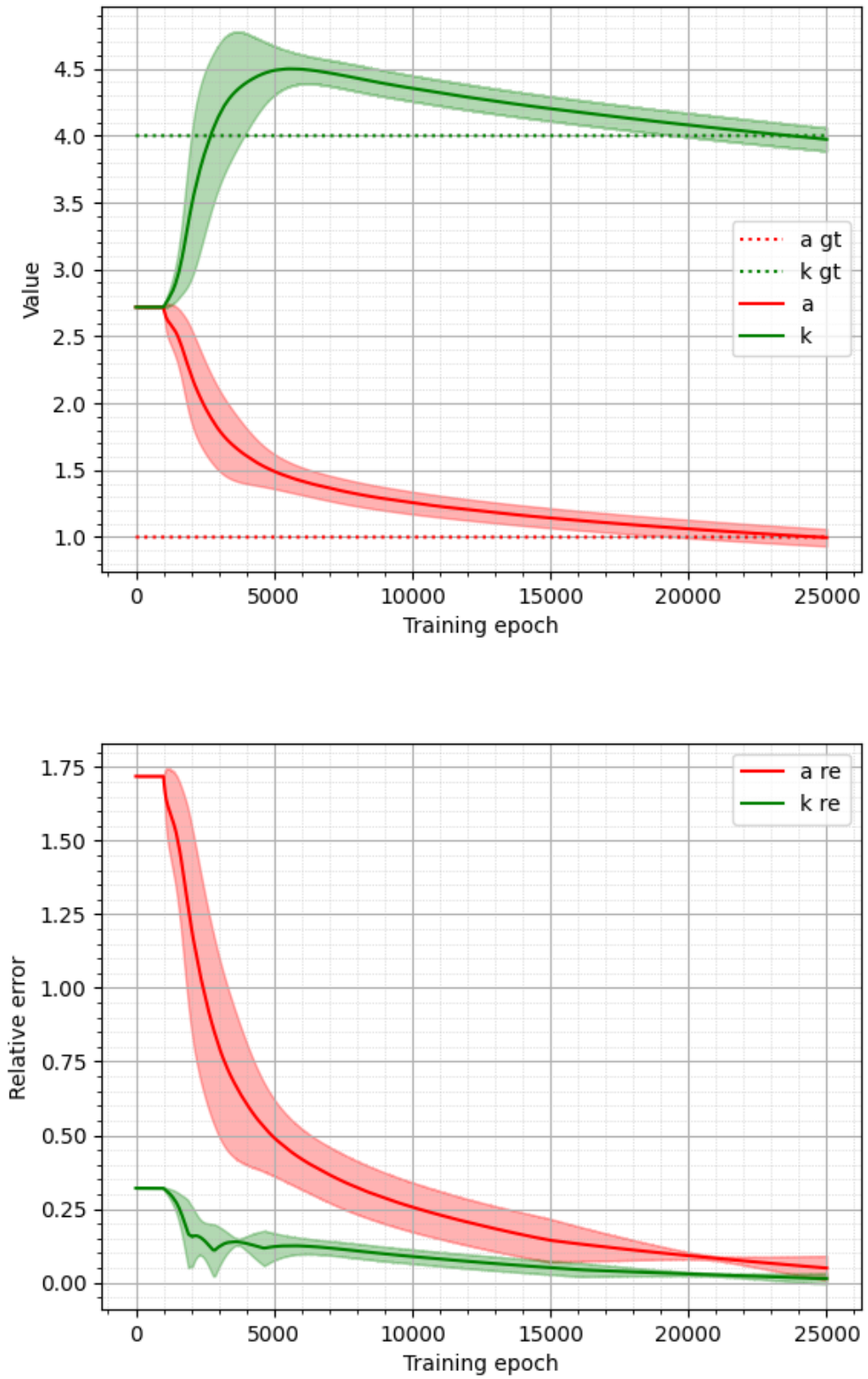

Figure 3.5. Mean and standard deviation of parameters a and k over 5 independent training runs. Both parameters are initialized at Euler's constant.

Training curves for parameters α_1, α_2, a and k can be observed on Figure 3.3, Figure 3.4 and Figure 3.5 respectively. Ground truth values are shown by dotted lines. Derivative orders α_1 and α_2 find the plateau first at 5000th training epoch, while linear parameters a and k converge more slowly with a only seeming to hit a plateau at the end of training.

3.2.3 Results with 10% noise

With a similar setup to the section, the synthetic data is now disturbed with 10% Gaussian noise. Shape of the input data during the last run can be seen on Figure 3.6, compared to the reference data on Figure 3.7. Medians of loss components over 5 independent runs can be observed on Figure 3.8. Compared to the result with 1% noise curve for the data loss now strictly follows the PDE loss.

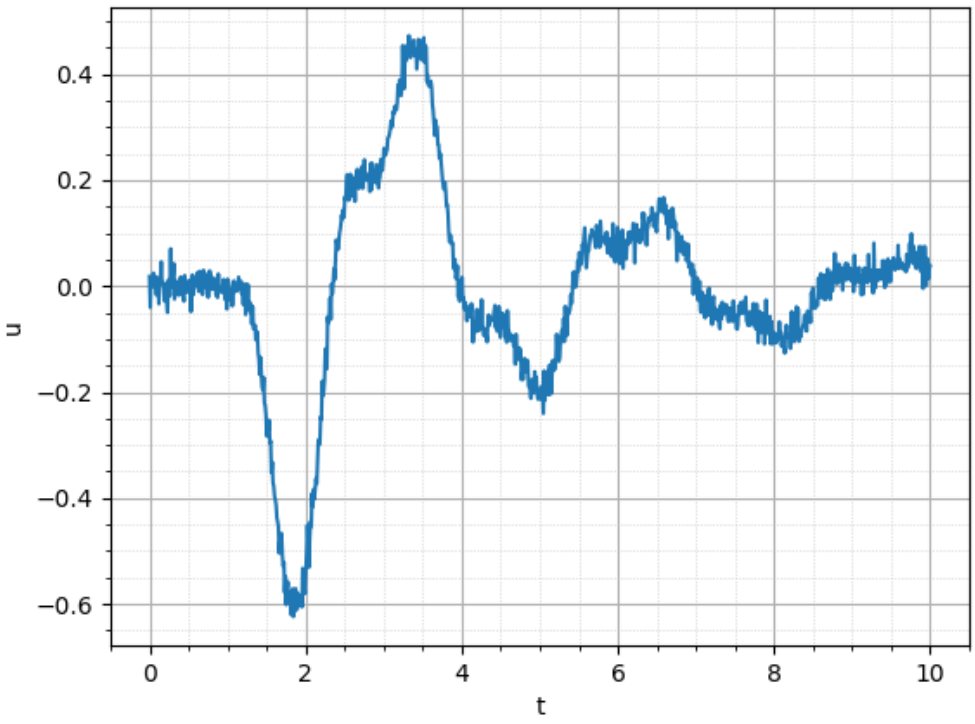

Figure 3.6. Synthetic data with 10% Gaussian noise

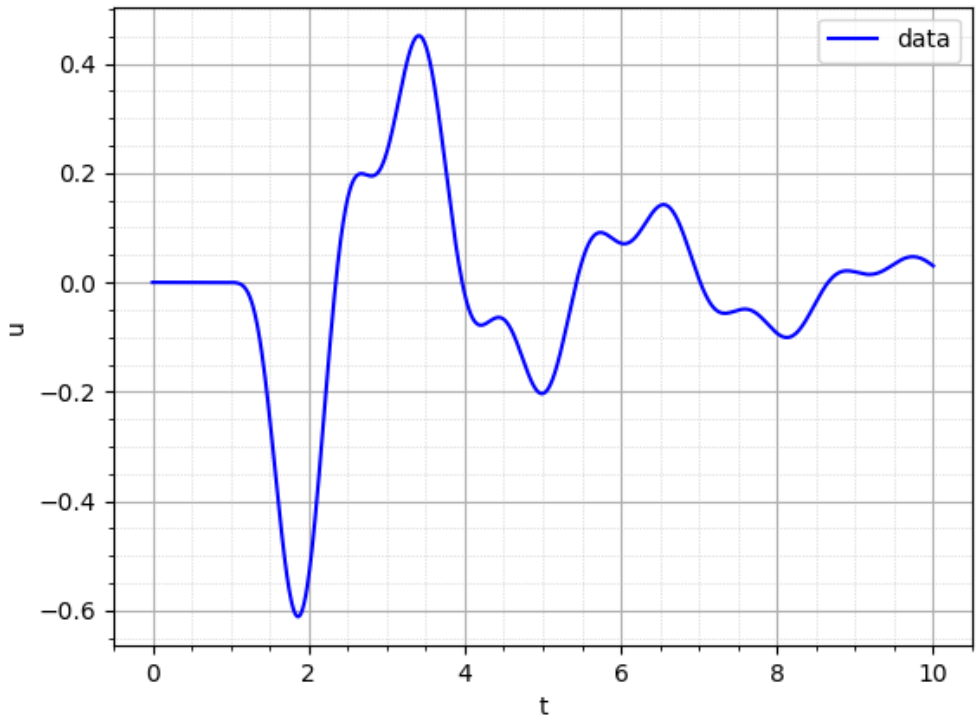

Figure 3.7. Reference data

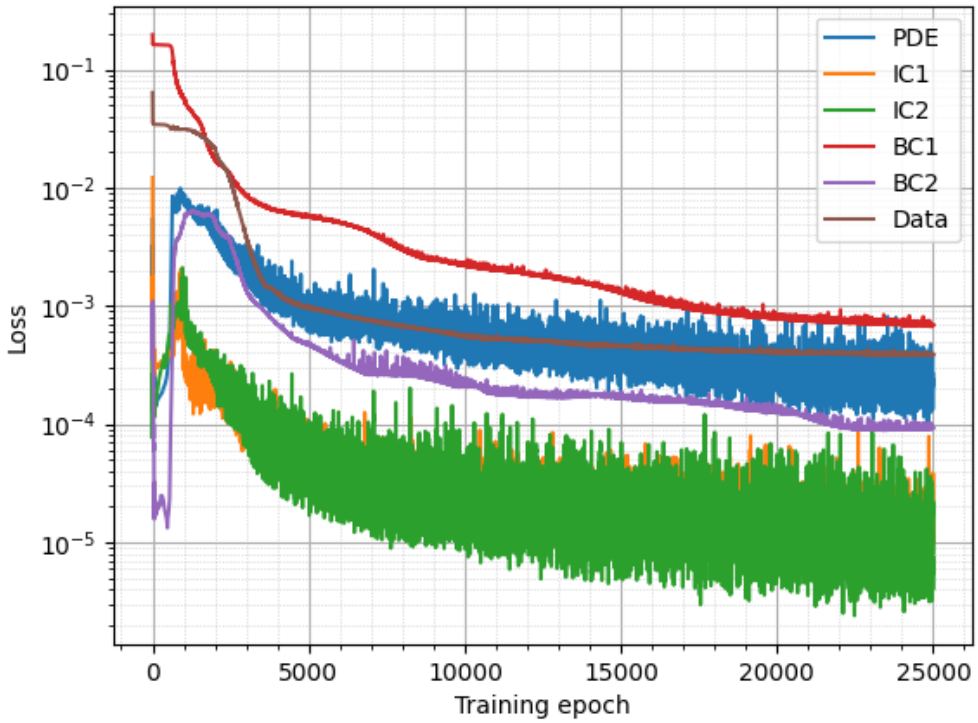

Figure 3.8. Median loss components over 5 independent training runs

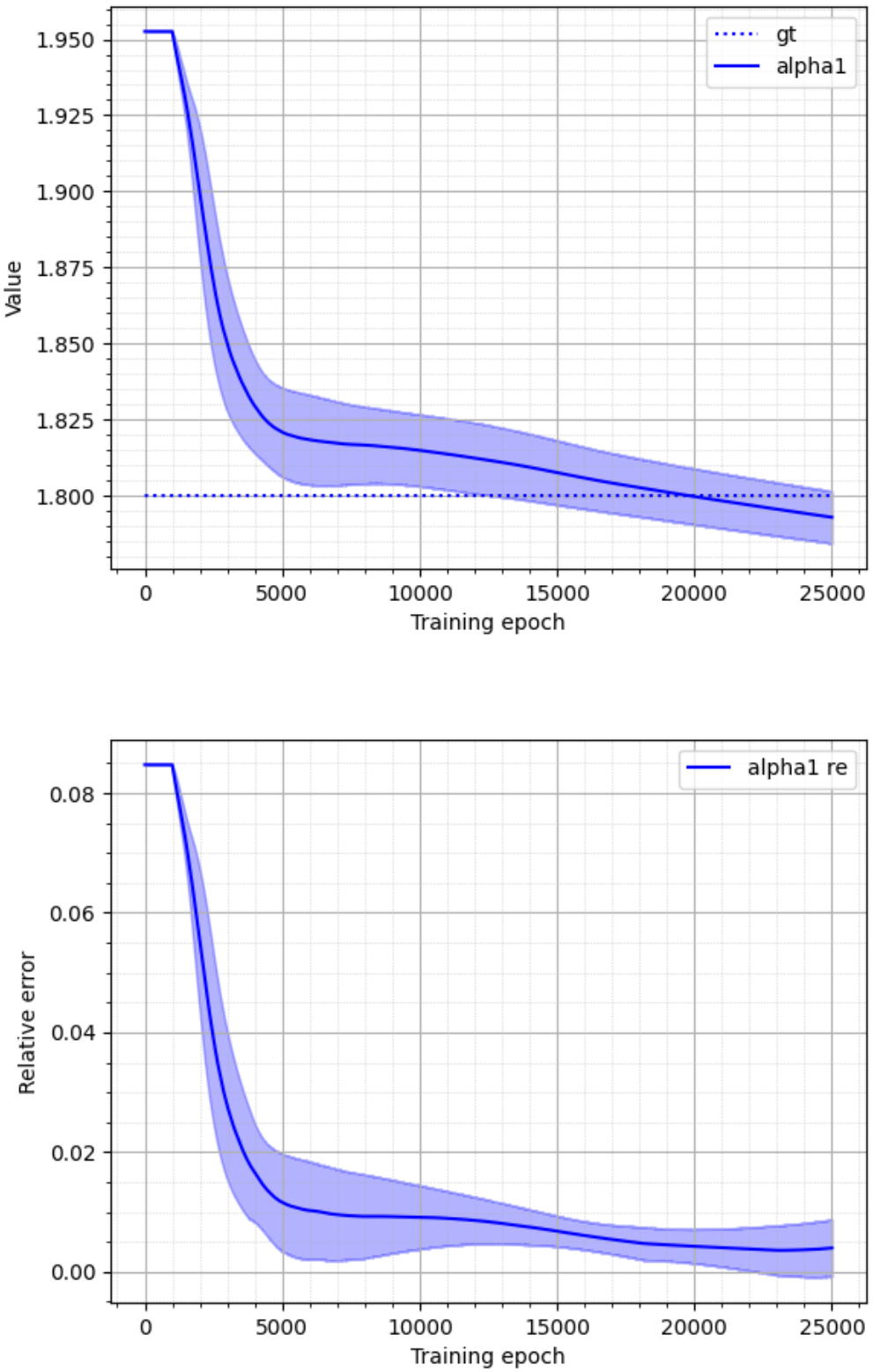

Figure 3.9. Mean and standard deviation of parameter α_1 over 5 independent training runs

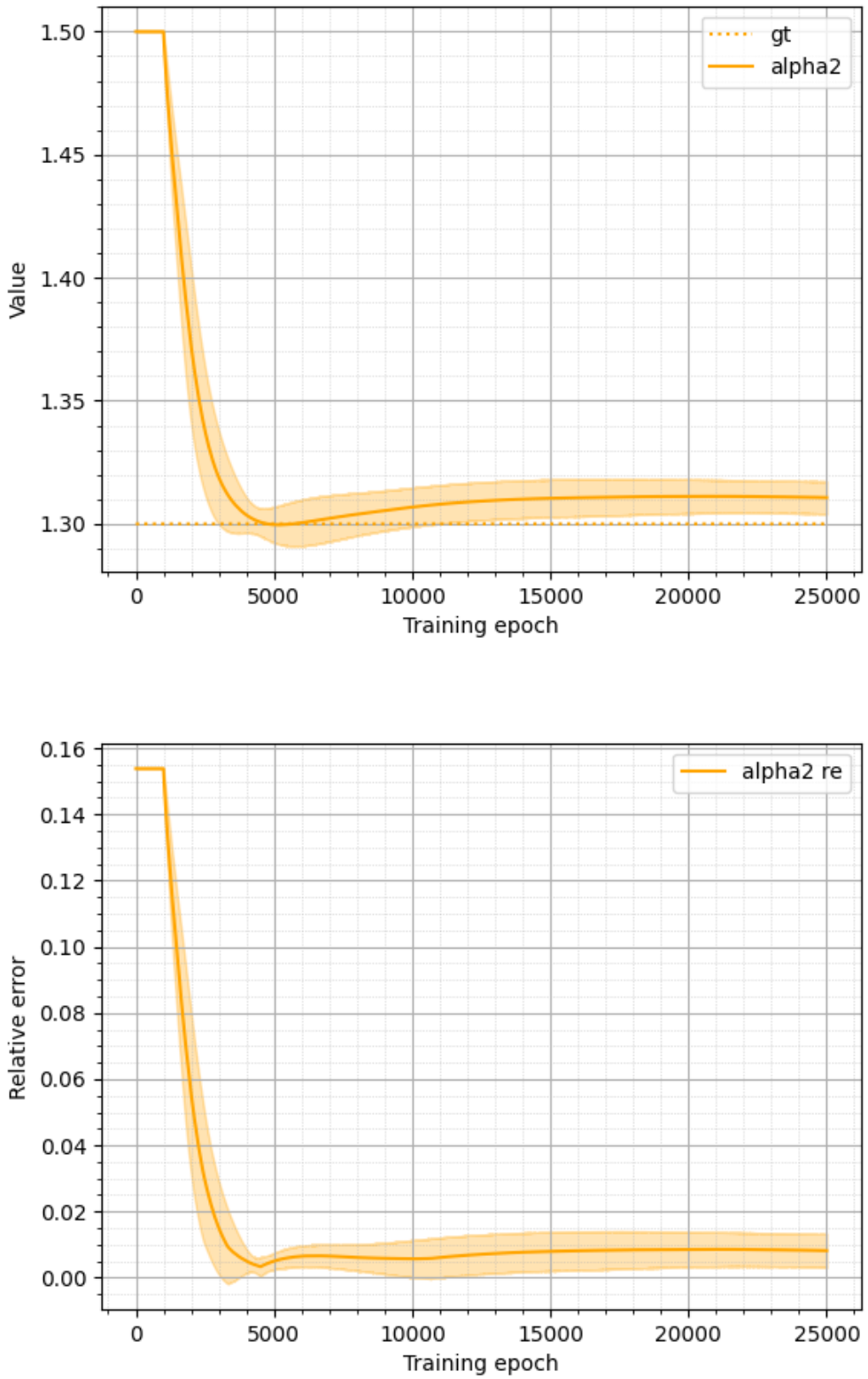

Figure 3.10. Mean and standard deviation of parameter α_2 over 5 independent training runs

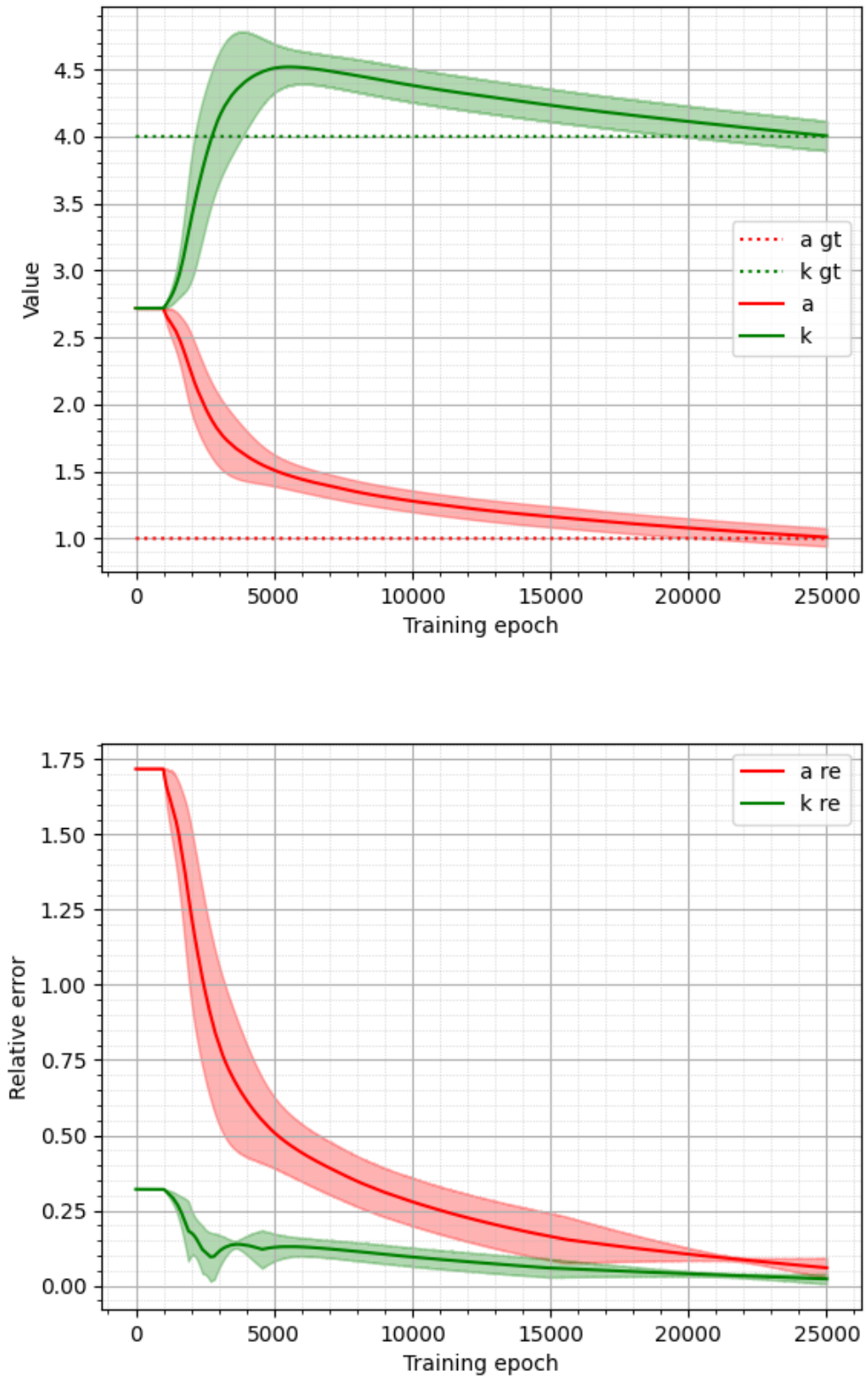

Figure 3.11. Mean and standard deviation of parameters a and k over 5 independent training runs. Both parameters are initialized at Euler's constant.

Figures 3.9, 3.10 and 3.11 show no significant changes in dynamics of the parameters, which means added noise did not have effect on the training process. Comparison of input data and network's prediction for the data at the end of the last run can be seen on Figure 3.12.

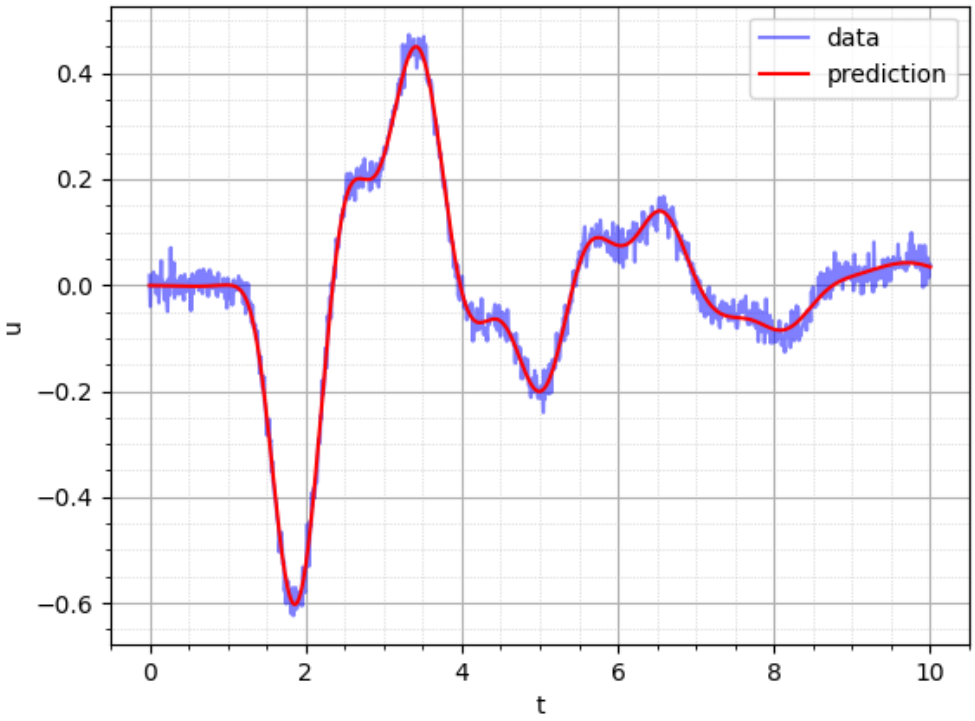

Figure 3.12. Noisy data and neural network's prediction

This suggests that imposition of the underlying PDE as well as healthy loss component balancing worked as a natural smoothing filter.

4. Inverse problem for time-fractional wave equation with point source

In this chapter we consider a time-fractional wave equation with a point source:

$$\begin{cases} {}^{C}\mathrm{D}^{\alpha_1}u(x,t) + a\,{}^{C}\mathrm{D}^{\alpha_2}u(x,t) - ku_{xx}(x,t) = A\delta(x-x_0)\mathcal{H}(t_0-t),\ x\in(0,l),\ t>0, \\ u(0,t)=0,\ u(l,t)=0,\ t>0, \\ u(x,0)=u_t(x,0)=0,\ x\in(0,l), \\ 1<\alpha_2<\alpha_2<2. \end{cases} \tag{4.0.1}$$

Here $\delta(x)$ is the Dirac's delta function and $\mathcal{H}(t)$ is the Heaviside step function. The forcing at $x = x_0$ is "switched on" at $t = 0$ and "switched off" at $t = t_0$.

4.1 Analysis of the Inverse problem

4.1.1 Eigenvalues and eigenfunctions of the Laplace operator with Dirichlet boundary conditions

We find eigenvalues and eigenfunctions for the second derivative subject to Dirichlet boundary conditions:

$$\begin{cases} \phi_{xx} = -\lambda\phi, \\ \phi(0)=\phi(l)=0. \end{cases} \tag{4.1.1}$$

Since (4.1.1) has non-trivial solutions only in the case $\lambda > 0$, we consider general solution in the form:

$$\phi(x) = A\cos(\sqrt{\lambda}x) + B\sin(\sqrt{\lambda}x). \tag{4.1.2}$$

Taking into account the Dirichlet condition at point $x = 0$:

$$\phi(0) = A = 0 \implies \phi(x) = B\sin(\sqrt{\lambda}x). \tag{4.1.3}$$

Similarly taking into account Dirihlet condition at the opposite edge $x = l$:

$$\phi(l) = B\sin(\sqrt{\lambda}l) = 0, \tag{4.1.4}$$

$$\sin(\sqrt{\lambda}l) = 0, \tag{4.1.5}$$

$$\sqrt{\lambda}l = \pi n,\ n \in \mathbb{N}. \tag{4.1.6}$$

With this equation (4.1.1) produces eigenvalues and respective eigenfunction:

$$\lambda_n = \left(\frac{\pi n}{l}\right)^2,\ n \in \mathbb{N}, \tag{4.1.7}$$

$$\phi_n(x) = B\sin\left(\frac{\pi n x}{l}\right). \tag{4.1.8}$$

4.1.2 Forward problem

Using the result from previous section, we expand $u(x,t)$ into the Fourier sine series:

$$u(x,t) = \sum_{n=1}^{\infty} u_n(t)\sin\left(\frac{\pi n x}{l}\right), \tag{4.1.9}$$

$$u_n(t) = \frac{2}{l}\int_0^l u(x,t)\sin\left(\frac{\pi n x}{l}\right). \tag{4.1.10}$$

Point source $\delta(x - x_0)$ has the series representation:

$$\delta(x - x_0) = \frac{2}{l}\sum_{n=1}^{\infty}\sin\left(\frac{\pi n x_0}{l}\right)\sin\left(\frac{\pi n x}{l}\right). \tag{4.1.11}$$

With this we expand all terms of equation (4.0.1) into the Fourier sine series:

$$\sum_{n=1}^{\infty}\left[{}^{C}\mathrm{D}^{\alpha_1}u_n(t)+{}^{C}\mathrm{D}^{\alpha_2}u_n(t)+k\frac{n^2\pi^2}{l^2}u_n(t)-A\frac{2}{l}\mathcal{H}(t_0-t)\sin\left(\frac{\pi n x_0}{l}\right)\right]\sin\left(\frac{n\pi x}{l}\right)=0, \tag{4.1.12}$$

$${}^{C}\mathrm{D}^{\alpha_1}u_n(t)+{}^{C}\mathrm{D}^{\alpha_2}u_n(t)+k\frac{n^2\pi^2}{l^2}u_n(t)-A\frac{2}{l}\mathcal{H}(t_0-t)\sin\left(\frac{\pi n x_0}{l}\right)=0. \tag{4.1.13}$$

Applying the Laplace transform yields an algebraic equation:

$$s^{\alpha_1}V_n(s)+as^{\alpha_2}V_n(s)+k\frac{n^2\pi^2}{l^2}V_n(s)-A\frac{2}{l}\sin\left(\frac{\pi n x_0}{l}\right)\frac{1}{s}(1-e^{-st_0})=0, \tag{4.1.14}$$

and solving for $V_n(s)$ gives us:

$$V_n(s)=\frac{A\frac{2}{l}\sin\left(\frac{\pi n x_0}{l}\right)(1-e^{-st_0})}{s^{\alpha_1+1}+as^{\alpha_2+1}+sk\lambda_n}. \tag{4.1.15}$$

Thus the solution to (4.0.1) in the complex-frequency domain takes form:

$$U(x,s)=\sum_{n=1}^{\infty}\frac{A\frac{2}{l}\sin\left(\frac{\pi n x_0}{l}\right)(1-e^{-st_0})}{s^{\alpha_1+1}+as^{\alpha_2+1}+sk\lambda_n}\sin\left(\frac{\pi n x}{l}\right). \tag{4.1.16}$$

4.1.3 Determining t_0

We have measurement data at point $x^*\in(0,l)$ and want to recover parameters a,k,α_1,α_2 (identifying the model) and at the same time parameters A,x_0,t_0 (identifying the source). Denoting:

$$\gamma_n=\sin\left(\frac{\pi n x^*}{l}\right), \tag{4.1.17}$$

we write down the solution (4.1.16) at $x=x^*$:

$$sU(x^*,s)\frac{l}{2}=\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)(1-e^{-st_0})}{s^{\alpha_1}+as^{\alpha_2}+k\lambda_n}\gamma_n. \tag{4.1.18}$$

Let $sU(x^*,s)\frac{2}{l} = A(s)$:

$$A(s) = \sum_{n=1}^{\infty} \frac{A \sin\left(\frac{\pi n x_0}{l}\right)(1-e^{-st_0})}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n}\gamma_n. \tag{4.1.19}$$

We compute the derivative of $A(s)$ with respect to s:

$$Q'(s) = \sum_{n=1}^{\infty} A\gamma_n \sin\left(\frac{\pi n x_0}{l}\right)\left(\frac{t_0 e^{-st_0}(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^2} - \frac{A(1-e^{-st_0})(\alpha_1 s^{\alpha_1 - 1} + \alpha_2 a s^{2\alpha_2 - 1})}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^2}\right). \tag{4.1.20}$$

In the limit process $s \to 0$:

$$\lim_{s\to 0} Q'(s) = \frac{At_0}{k}\sum_{n=1}^{\infty}\frac{\gamma_n}{\lambda_n}\sin\left(\frac{\pi n x_0}{l}\right), \tag{4.1.21}$$

and the resulting series converges absolutely since $\lambda_n \sim n^2$ and $|\gamma_n| \leq 1$. We differentiate with respect to s once again:

$$Q''(s) = -\sum_{n=1}^{\infty} A\gamma_n \sin\left(\frac{\pi n x_0}{l}\right)\left(\frac{t_0^2 e^{-st_0}(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^3}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^4}\right) + o(1). \tag{4.1.22}$$

In the limit $s \to 0$:

$$\lim_{s\to 0} Q''(s) = -\frac{At_0^2}{k}\sum_{n=1}^{\infty}\frac{\gamma_n}{\lambda_n}\sin\left(\frac{\pi n x_0}{l}\right). \tag{4.1.23}$$

From (4.1.21) and (4.1.23) it follows that:

$$t_0 = -\lim_{s\to 0}\frac{Q''(s)}{Q'(s)}, \tag{4.1.24}$$

and t_0 is uniquely defined.

4.1.4 Determining α_2

We return to:

$$\frac{A(s)}{(1-e^{-st_0})} = \sum_{n=1}^{\infty} \frac{A\sin\left(\frac{\pi n x_0}{l}\right)}{s^{\alpha_1} + as^{\alpha_2} + k\lambda_n}\gamma_n. \tag{4.1.25}$$

If we compute the derivative with respect to s and factor out s^{α_2}:

$$\left(\frac{A(s)}{(1-e^{-st_0})}\right)' = -s^{\alpha_2-1}\sum_{n=1}^{\infty} \frac{A\sin\left(\frac{\pi n x_0}{l}\right)(\alpha_1 s^{\alpha_1-\alpha_2} + 2a\alpha_2)}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^2}\gamma_n. \tag{4.1.26}$$

Using the natural logarithm we can find an expression for $\alpha_2 - 1$:

$$\left|\left(\frac{A(s)}{(1-e^{-st_0})}\right)'\right| = |s|^{\alpha_2-1}\left|\sum_{n=1}^{\infty} \frac{A\sin\left(\frac{\pi n x_0}{l}\right)(\alpha_1 s^{\alpha_1-\alpha_2} + a\alpha_2)}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^2}\gamma_n\right|, \tag{4.1.27}$$

$$\ln\left|\left(\frac{A(s)}{(1-e^{-st_0})}\right)'\right| = (\alpha_2-1)\ln|s| + \ln\left|\sum_{n=1}^{\infty} \frac{A\sin\left(\frac{\pi n x_0}{l}\right)(\alpha_1 s^{\alpha_1-\alpha_2} + a\alpha_2)}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^2}\gamma_n\right|, \tag{4.1.28}$$

$$\alpha_2 - 1 = \frac{\ln\left|\left(\frac{A(s)}{(1-e^{-st_0})}\right)'\right|}{\ln|s|} - \frac{\ln\left|\sum_{n=1}^{\infty} \frac{A\sin\left(\frac{\pi n x_0}{l}\right)(\alpha_1 s^{\alpha_1-\alpha_2} + a\alpha_2)}{(s^{\alpha_1} + as^{\alpha_2} + k\lambda_n)^2}\gamma_n\right|}{\ln|s|}. \tag{4.1.29}$$

In the limit $s \to 0$:

$$\alpha_2 - 1 = \lim_{s\to 0} \frac{\ln\left|\left(\frac{A(s)}{(1-e^{-st_0})}\right)'\right|}{\ln|s|}, \tag{4.1.30}$$

and α_2 is uniquely defined.

4.1.5 Determining α_1 and a

We apply further transformations:

$$-s^{1-\alpha_2}\left(\frac{A(s)}{(1-e^{-st_0})}\right)' = \sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)(\alpha_1 s^{\alpha_1-\alpha_2}+a\alpha_2)}{(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^2}\gamma_n, \tag{4.1.31}$$

$$\lim_{s\to 0} -s^{1-2\alpha_2}\left(\frac{A(s)}{(1-e^{-st_0})}\right)' = \frac{Aa\alpha_2}{c^4}\sum_{n=1}^{\infty}\frac{\gamma_n}{\lambda_n^2}\sin\left(\frac{\pi n x_0}{l}\right), \tag{4.1.32}$$

and the structure on the right hand side is uniquely defined. We denote:

$$B(s) = -s^{1-\alpha_2}\left(\frac{A(s)}{(1-e^{-st_0})}\right)'. \tag{4.1.33}$$

We differentiate $B(s)$ and find an expression to α_1 in a similar fashion to α_2:

$$B'(s) = s^{\alpha_1-\alpha_2-1}\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)\alpha_1(\alpha_1-\alpha_2)(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^2}{(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^4}\gamma_n + o(1), \tag{4.1.34}$$

$$|B'(s)| = |s|^{\alpha_1-\alpha_2-1}\left|\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)\alpha_1(\alpha_1-\alpha_2)(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^2}{(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^4}\gamma_n + o(1)\right|, \tag{4.1.35}$$

$$\ln|B'(s)| = (\alpha_1-\alpha_2-1)\ln|s| + \ln|O(1)+o(1)|. \tag{4.1.36}$$

In the limit $s\to 0$:

$$(\alpha_1-\alpha_2-1) = \lim_{s\to 0}\frac{\ln|B'(s)|}{\ln|s|}, \tag{4.1.37}$$

and α_1 is defined uniquely. From (4.1.34):

$$\lim_{s\to 0} B'(s)s^{-\alpha_1+\alpha_2+1} = \sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)\alpha_1(\alpha_1-\alpha_2)(c^4\lambda_n^2)}{c^8\lambda_n^4}\gamma_n, \tag{4.1.38}$$

$$\frac{1}{\alpha_1(\alpha_1-\alpha_2)}\lim_{s\to 0} B'(s)s^{-\alpha_1+\alpha_2+1} = \frac{A}{c^4}\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)}{\lambda_n^2}\gamma_n. \tag{4.1.39}$$

We can divide (4.1.39) by (4.1.32) and find out that a is uniquely defined.

4.1.6 Determining A and x_0

Returning back to $B(s)$:

$$\frac{B(s)}{(\alpha_1 s^{\alpha_1-\alpha_2}+a\alpha_2)}=\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)}{(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^2}\gamma_n, \tag{4.1.40}$$

we denote left-hand side as $C(s)$ and once again differentiate with respect to s:

$$C'(s)=-2\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)(\alpha_1 s^{\alpha_1-1}+\alpha_2 a s^{\alpha_2-1})}{(s^{\alpha_1}+as^{\alpha_2}+k\lambda_n)^3}\gamma_n, \tag{4.1.41}$$

$$-\frac{C'(s)}{2(2\alpha_1 s^{2\alpha_1-1}+2\alpha_2 a s^{2\alpha_2-1})}=\sum_{n=1}^{\infty}\frac{A\sin\left(\frac{\pi n x_0}{l}\right)}{(s^{2\alpha_1}+as^{2\alpha_2}+k\lambda_n)^3}\gamma_n. \tag{4.1.42}$$

This can be continued indefinitely. Considering that $\lambda_n \sim n^2$, in the limit $s \to 0$ we have a uniquely defined sequence:

$$V_i=\frac{A}{k^i}\sum_{n=1}^{\infty}\frac{\gamma_n}{n^{2i}}\sin\left(\frac{\pi n x_0}{l}\right), \quad i\in\mathbb{N}. \tag{4.1.43}$$

Advancing the sequence one step further, we get:

$$V_{i+1}=\frac{A}{k^{(i+1)}}\sum_{n=1}^{\infty}\frac{\gamma_n}{n^{2(i+1)}}\sin\left(\frac{\pi n x_0}{l}\right). \tag{4.1.44}$$

Dividing V_i by V_{i+1} we get a sequence:

$$\frac{V_i}{V_{i+1}}=k\frac{\sum_{n=1}^{\infty}\frac{\gamma_n}{n^{2i}}\sin\left(\frac{\pi n x_0}{l}\right)}{\sum_{n=1}^{\infty}\frac{\gamma_n}{n^{2(i+1)}}\sin\left(\frac{\pi n x_0}{l}\right)}=k\frac{\left(\gamma_1\sin\left(\frac{\pi x_0}{l}\right)+O(n^{-2(i+1)})\right)}{\left(\gamma_1\sin\left(\frac{\pi x_0}{l}\right)+O(n^{-2i})\right)}, \tag{4.1.45}$$

$$\lim_{i\to\infty}\frac{V_i}{V_{i+1}}=k, \tag{4.1.46}$$

and k on is uniquely defined. Returning to the sequence V_i:

$$\frac{V_i}{k^i} = A\sum_{n=1}^{\infty} \frac{\gamma_n}{n^{2i}} \sin\left(\frac{\pi n x_0}{l}\right) = A\gamma_1 \sin\left(\frac{\pi x_0}{l}\right) + O(n^{-2i}). \tag{4.1.47}$$

In the limit $i \to \infty$:

$$\lim_{i\to\infty} \frac{V_i}{k^i} = A\gamma_1 \sin\left(\frac{\pi x_0}{l}\right), \tag{4.1.48}$$

$$A = \frac{\gamma_1}{\sin\left(\frac{\pi x_0}{l}\right)} \lim_{i\to\infty} \frac{V_i}{k^i}. \tag{4.1.49}$$

As a result we have an uniquely defined function:

$$f_i(x) = \sum_{n=1}^{\infty} \frac{\gamma_n}{n^{2i}} \frac{\sin(nx)}{\sin(x)}, \tag{4.1.50}$$

where $x = \frac{\pi x_0}{l}$ and $x_0 \in (0, l)$. Expanding the sum yields:

$$f_i(x) = \gamma_1 + \frac{\gamma_2}{4^i} \frac{\sin(2x)}{\sin(x)} + \frac{\gamma_3}{9^i} \frac{\sin(3x)}{\sin(x)} + ..., \tag{4.1.51}$$

$$f_i(x) - \gamma_1 = \frac{\gamma_2}{4^k} \frac{\sin(2x)}{\sin(x)} + \frac{\gamma_3}{9^k} \frac{\sin(3x)}{\sin(x)} + ..., \tag{4.1.52}$$

$$4^i(f_i(x) - \gamma_1) = \gamma_2 \frac{\sin(2x)}{\sin(x)} + \gamma_3 \left(\frac{4}{9}\right)^i \frac{\sin(3x)}{\sin(x)} + ... \tag{4.1.53}$$

In the limit $i \to \infty$:

$$\lim_{i\to\infty} \frac{4^i}{\gamma_2}(f_i(x) - \gamma_1) = \frac{\sin(2x)}{\sin(x)} = 2\cos(x). \tag{4.1.54}$$

Since $\cos(x)$ is invertible for $x \in (0, \pi)$, x_0 is uniquely defined. From (4.1.49) it follows that A is uniquely defined.

4.2 Numerical experiments

For the numerical experiment we consider an equation:

$$\begin{cases} {}^{C}\mathrm{D}^{1.9}u(x,t) + 1.4\,{}^{C}\mathrm{D}^{1.6}u(x,t) - 3u_{xx}(x,t) = 10\delta(x-0.4)\mathcal{H}(4-t) \;\; x \in (0,1),\;\; t > 0, \\ u(0,t) = 0,\;\; u(1,t) = 0,\;\; t > 0, \\ u(x,0) = u_t(x,0) = 0,\;\; x \in (0,1). \end{cases} \tag{4.2.1}$$

In order to deal with the Dirac's delta function within PINN framework, method proposed in [39] suggests using a smooth approximation with a Gaussian curve:

$$\tilde{\delta}(x - x_0) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-x_0)^2}{2\sigma^2}}, \tag{4.2.2}$$

where σ is small enough. In addition to that computational domain is divided into two parts: $\Omega = \Omega_{r_0} \cup \Omega_{r_1}$ such that:

$$\Omega_{r_0} = \{x_0 + x \in \Omega \mid |x| \leq 3\sigma\},\;\; \Omega_{r_1} = \Omega \setminus \Omega_{r_0}. \tag{4.2.3}$$

With collocation points sampled in Ω_{r_0} and Ω_{r_1}, the residual loss is correspondingly divided into two terms:

$$\mathrm{L}_r = \lambda_{r_0}\mathrm{L}_{r_0} + \lambda_{r_1}\mathrm{L}_{r_1}. \tag{4.2.4}$$

Heaviside function can be smoothly approximated using logistic sigmoid parametrized by η:

$$\tilde{\mathcal{H}}(t) = \frac{1}{1 + e^{-\eta t}}. \tag{4.2.5}$$

In order to tune the coefficients λ_i, a trainable parameter σ_i and higher bound κ are defined such that:

$$\lambda_i = \frac{1}{\kappa^{-1} + \sigma_i^2}, \tag{4.2.6}$$

and with that an additional penalty term is added to the total loss function:

$$\mathrm{L}_\lambda = \sum_{i=1}^{m} \log(\lambda_i^{-1}). \tag{4.2.7}$$

With that the weights can be learned by an optimizer instance.

Applying numerical inverse Laplace transform to (4.1.16) produces the reference solution that can be seen on Figure 4.1. With source term variables frozen at ground truth values $x_0 = 0.4$, $t_0 = 4.0$ and $A = 6.0$, recovery of parameters α_1, α_2, a and k was attempted using hyper-parameter values in Table 2 and methods described in this section.

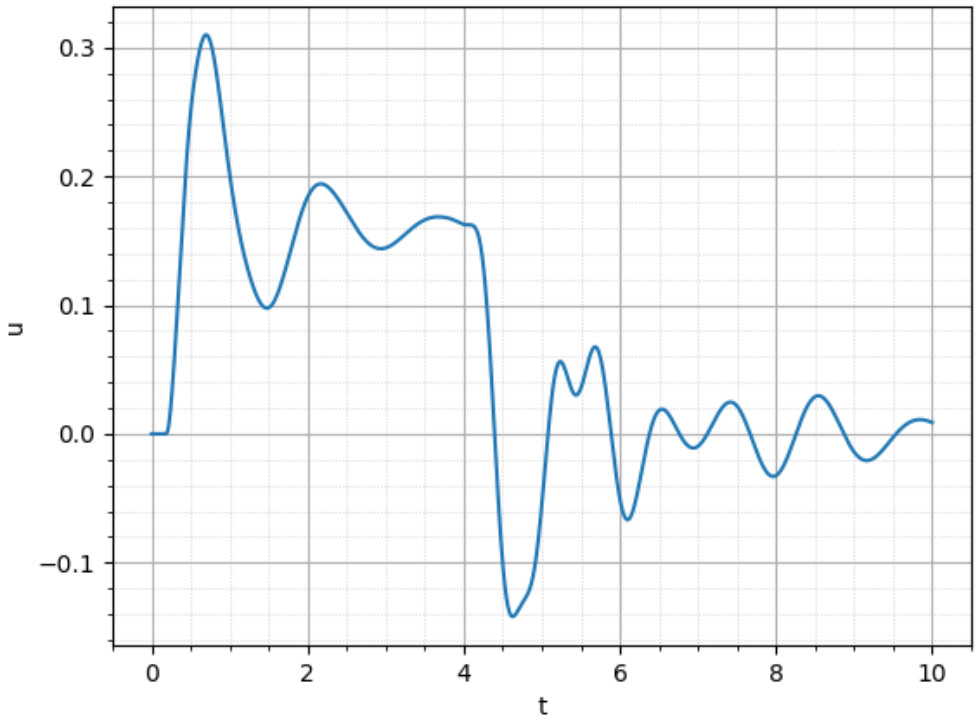

Figure 4.1. Generated data at $x = x^*$

D	3
W	128
$N_{\Omega_{r_0}}$	1000
$N_{\Omega_{r_1}}$	2500
$N_{\partial\Omega}$	500
N_{Ω_0}	500
$\eta_{\mathbf{w}}$	10^{-3}
η_{pars}	10^{-3}
σ	10^{-2}
η	25

Table 2. All training hyperparameters and chosen values for point source problem

Resulting losses after 5000 training epochs can be seen on Figure 4.2. Both residual losses (representing samples at Ω_1 and Ω_0 respectively) plateau at step 2000 alongside, indicating failure to converge or convergence to a local optimum. Network's prediction for the data points can be seen on Figure 4.3, which shows model's failure the replicate oscillatory behavior. Without the solution forming an anchor the optimizer tries to compensate losses by drifting away the parameters, which can be seen for α_1 on Figure 4.4.

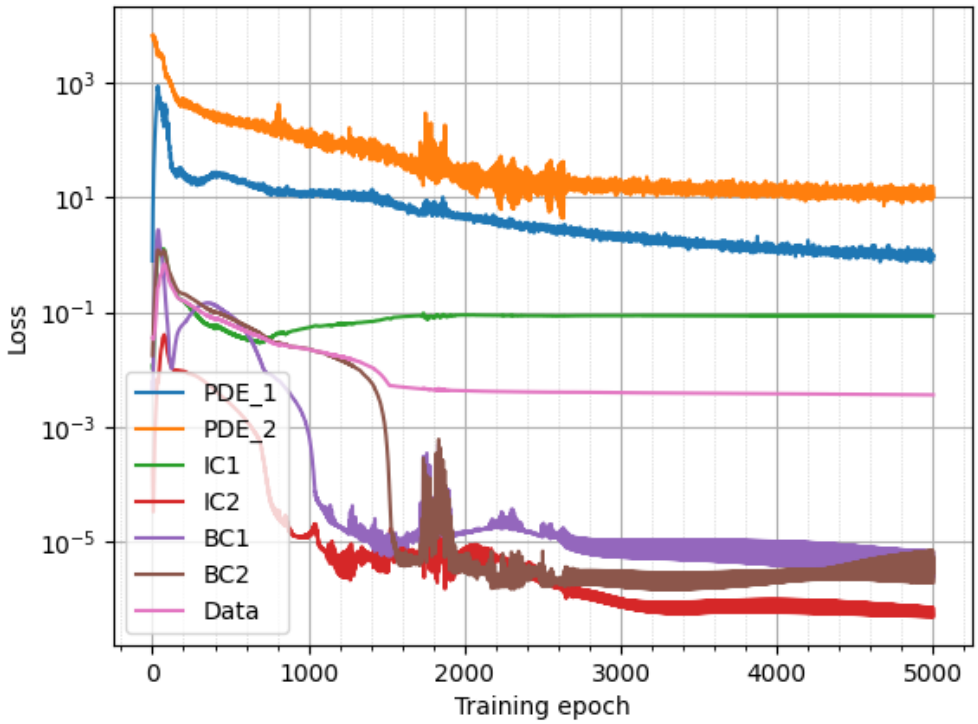

Figure 4.2. Loss components

This might show classical PINN's inability to handle sharp shocks and transitions (like at $t = t_0$ and $x = x_0$). A common drawback specific to classical PINNs that might be relevant

here is the spectral bias - neural networks tend to rapidly learn low frequency components (for example in the solution and struggle (or ignore) higher frequency components [40]. Another possibility is simple lack of network complexity, and solving a problem like this (point source + non-local operators) requires scaling width and depth beyond the current memory constraints.

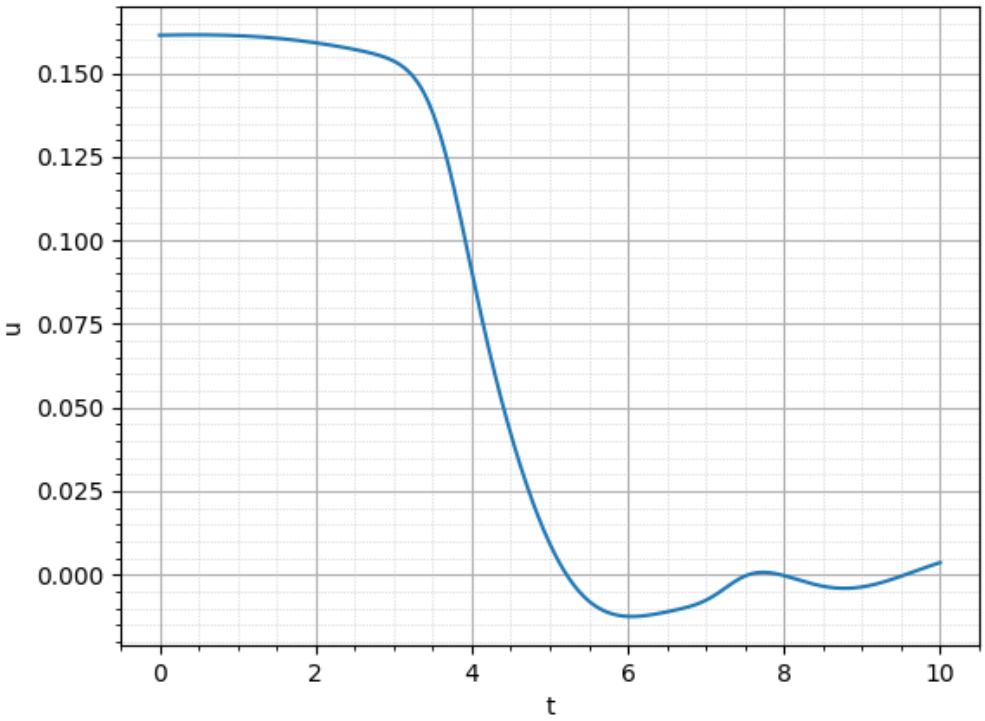

Figure 4.3. Model's prediction at $x = x^*$

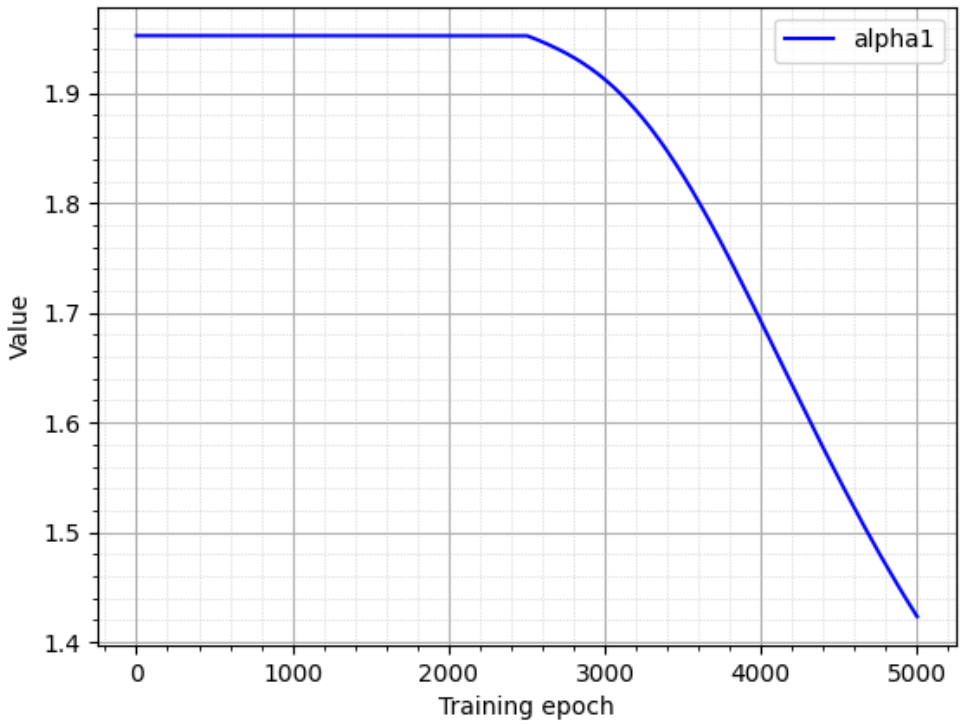

Figure 4.4. Drift of parameter α_1 towards 1.

5. Summary

Throughout this master's thesis, inverse problems for time-fractional wave equations with two Caputo fractional derivatives were studied. Laplace transform and Fourier methods were used to construct analytical solutions to an equation with mixed boundary conditions and an equation with a point source, and uniqueness of parameter recovery was shown for both equations. Utilizing Gauss-Jacobi quadrature to approximate the Caputo fractional derivative, an attempt was made at solving the inverse problems using physics informed neural networks. PINN solver for time-fractional wave equation with mixed boundary conditions showed good predictions for the inferred parameters and robustness to noise. On the other hand, it completely failed to replicate the oscillatory behavior of the time-fractional wave equation with point source. This might point at inability of classical PINNs to handle non-smooth problems, or lack of complexity of the neural network within allowed memory constraints.

Further research on this topic could consider solving point source problem for time-fractional wave equation utilizing hybrid PINN + weighted residual approach where the residual loss is integrated against a family of test functions. Another interesting field to go forward with is application of complex-valued neural networks to inverse problems studied in this thesis. Using the Laplace transform does not change the equation parameters, but effectively turns the Caputo derivative into multiplication in the complex-frequency domain.

Acknowledgements

I would like to express my deepest gratitude to supervisor Jaan Janno for invaluable guidance and help throughout the creation of this thesis.

I would also like to sincerely thank all the lecturers in the Applied Physics study programme for their support, fair treatment and sheer dedication to passing knowledge to new generations.

And of course thank you to the reader for taking the time with this work - your attention is sincerely appreciated.

References

[1] Mihailo P. Lazarević, Milan Rade Rapaić, Tomislav B. Šekara *Introduction to Fractional Calculus with Brief Historical Background* WSEAS Press, 2014

[2] Francesco Mainardi *Fractional calculus and waves in linear viscoelasticity : an introduction to mathematical models* World Scientific, 2022

[3] Inés Tejado, Emiliano Hernández and Duarte Valério *Economic growth in the European Union modelled with fractional derivatives: first results* Bulletin of the Polish Academy of Sciences, Technical Sciences, Vol. 66, No. 4, 2018 DOI: `https://doi.org/10.24425/124262`

[4] Joshua Kiddy K. Asamoah, Isaac K. Adu, Fredrick A. Wireko, Adu Sakyi *Mathematical Modeling of Giardiasis Transmission Dynamics Using Caputo Fractional Derivative* Engineering Reports, Vol. 8, No. 3, 2026 DOI: `https://doi.org/10.1002/eng2.70664`

[5] Azroul, E., Kamali, N., Shimi, M *Novel insights into Cassava mosaic disease using Caputo fractional derivative: modeling and analysis* Applicable Analysis , Vol. 105, No. 1, 2026 DOI: `https://doi.org/10.1080/00036811.2025.2509314`

[6] Mohammed Alabedalhadi, Wael Salameh, Shrideh Al-Omari, Yeliz Karaca, Mohammed Al-Smadi, Dumitru Baleanu, and Shaher Momani *Analysis of a fractional order model for the interaction between growth and crowding of tumor step cells via Caputo fractional derivatives* Fractals, 2026 DOI: `https://doi.org/10.1142/S0218348X26400128`

[7] Charles W. Groetsch, *Inverse Problems in The Mathematical Sciences* Springer, 1993

[8] M. Raissi, P. Perdikaris, G.E. Karniadakis *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations* Journal of Computational Physics, Vol. 378, 2019 DOI: `https://doi.org/10.1016/j.jcp.2018.10.045`

[9] Weisstein, Eric W., *Laplace Transform* From MathWorld–A Wolfram Web Resource. URL: `https://mathworld.wolfram.com/LaplaceTransform.html`

[10] Guillermo Navas-Palencia, *Faster numerical inverse Laplace transform in mpmath* URL: `http://gnpalencia.org/blog/2022/invertlaplace/`

[11] Alar Leibak, *Kompleksmuutuja Funktsioonid* TTÜ Kirjastus, 2015

[12] Kai Diethelm, *The Analysis of Fractional Differential Equations* Springer, 2010

[13] Trifce Sandev, Živorad Tomovski *Fractional Equations and Models* Springer, 2019

[14] Tamara G Grossmann, Urszula Julia Komorowska, Jonas Latz, Carola-Bibiane Schönlieb *Can physics-informed neural networks beat the finite element method?* IMA Journal of Applied Mathematics, Vol. 89, No. 1, 2024 DOI: `https://doi.org/10.1093/imamat/hxae011`

[15] Aleksandra Jekic, Afroditi Natsaridou, Signe Riemer-Sørensen, Helge Langseth, Odd Erik Gundersen *Examining the robustness of Physics-Informed Neural Networks to noise for Inverse Problems* arXiv pre-print: https://doi.org/10.48550/arXiv.2509.20191

[16] Tensorflow https://www.tensorflow.org/

[17] PyTorch https://pytorch.org/

[18] JAX https://docs.jax.dev/en/latest/

[19] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind *Automatic differentiation in machine learning: a survey* The Journal of Machine Learning Research, Vol. 18, No. 1, 2017 DOI: https://dl.acm.org/doi/10.5555/3122009.3242010

[20] Tensorflow API: Gradient tapes URL: https://www.tensorflow.org/api_docs/python/tf/GradientTape

[21] Shiv Ram Dubey, Satish Kumar Singh, Bidyut Baran Chaudhuri *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark* Neurocomputing, Vol. 503, 2022 DOI: https://doi.org/10.1016/j.neucom.2022.06.111

[22] Midhun T. Augustine *A survey on universal approximation theorems* arXiv pre-print: https://doi.org/10.48550/arXiv.2407.12895

[23] Sifan Wang, Yujun Teng, Paris Perdikaris *Understanding and mitigating gradient pathologies in physics-informed neural networks* SIAM Journal on Scientific Computing, Vol. 43, No. 5, 2021 DOI: https://doi.org/10.1137/20M1318043

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Deep Residual Learning for Image Recognition* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 DOI: https://doi.org/10.1109/CVPR.2016.90

[25] Jaan Janno, *Arvutusmeetodid* TTÜ Kirjastus, 2016

[26] Rafael Bischof, Michael A. Kraus *Multi-Objective Loss Balancing for Physics-Informed Deep Learning* Computer Methods in Applied Mechanics and Engineering, Vol. 439, 2025 DOI: https://doi.org/10.1016/j.cma.2025.117914

[27] Shota Deguchi and Mitsuteru Asai *Dynamic and norm-based weights to normalize imbalance in back-propagated gradients of physics-informed neural networks* Journal of Physics Communications, Vol. 7, No. 7, 2023 DOI: https://doi.org/10.1088/2399-6528/ace416

[28] Pancheng Niu, Yongming Chen, Jun Guo, Yuqian Zhou, Minfu Feng, Yanchao Shi *Improved physics-informed neural network in mitigating gradient-related failures* Neurocomputing, Vol. 638 DOI: https://doi.org/10.1016/j.neucom.2025.130167

[29] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, Andrew Rabinovich *GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks* Proceedings of the 35th International Conference on Machine Learning, 2018 arXiv: https://doi.org/10.48550/arXiv.1711.02257

[30] Shota Deguchi and Mitsuteru Asai *Reliable and efficient inverse analysis using physics-informed neural networks with normalized distance functions and adaptive weight tuning* Machine Learning: Science and Technology, Vol. 6, No. 4, 2025 DOI: https://doi.org/10.1088/2632-2153/ae1b71

[31] Diederik P. Kingma, Jimmy Ba *Adam: A Method for Stochastic Optimization* 3rd International Conference for Learning Representations, San Diego, 2015 arXiv: https://doi.org/10.48550/arXiv.1412.6980

[32] Ilya Loshchilov, Frank Hutter *Fixing Weight Decay Regularization in Adam* arXiv pre-print: https://doi.org/10.48550/arXiv.1711.05101

[33] Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, Madeleine Udell *Challenges in Training PINNs: A Loss Landscape Perspective* ICML'24: Proceedings of the 41st International Conference on Machine Learning, 2024 DOI: https://dl.acm.org/doi/10.5555/3692070.3693785

[34] Jing Li, Zhengqi Zhang *Transformed Diffusion-Wave fPINNs: Enhancing Computing Efficiency for PINNs Solving Time-Fractional Diffusion-Wave Equations* arXiv pre-print: https://doi.org/10.48550/arXiv.2506.11518

[35] SciPy API Reference https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.roots_jacobi.html

[36] Xiong-Bin Yan, Zhi-Qin John Xu, Zheng Ma *Laplace-fPINNs: Laplace-based fractional physics-informed neural networks for solving forward and inverse problems of subdiffusion* arXiv pre-print: https://doi.org/10.48550/arXiv.2304.00909

[37] Rayyan Abdalla *Complex-valued Neural Networks – Theory and Analysis* arXiv pre-print: https://doi.org/10.48550/arXiv.2312.06087

[38] Weisstein, Eric W., *Jacobi Theta Functions* From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/JacobiThetaFunctions.html

[39] Xiang Huang, Hongsheng Liu, Beiji Shi, Zidong Wang, Kang Yang, Yang Li, Bingya Weng, Min Wang, Haotian Chu, Jing Zhou, Fan Yu, Bei Hua, Lei Chen, Bin Dong *Solving Partial Differential Equations with Point Source Based on Physics-Informed Neural Networks* arXiv pre-print: https://doi.org/10.48550/arXiv.2111.01394

[40] Siavash Khodakaramia, Vivek Oommenb, Nazanin Ahmadi Daryakenarib, Maxim Beekenkamp, George Em Karniadakis *Spectral bias in physics-informed and operator learning: Analysis and mitigation guidelines* arXiv pre-print: https://doi.org/10.48550/arXiv.2602.19265

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Denis Akimov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Inverse problems for time-fractional wave equation", supervised by Jaan Janno
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

19.05.2026

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Code Examples

Python/Tensorflow code written for numerical sections of this thesis are available on Github repository: `https://github.com/DenAkimov/pinn-tfwaves`

Computations were made on a gaming workstation equipped with NVIDIA GeForce RTX 5060 GPU with 16 GB VRAM.

Example 1: Computation of second spatial derivative of the network using gradient tapes

```
@tf.function
def _laplacian(self, x, t):
 with tf.GradientTape() as grad2:
      grad2.watch(x)
      with tf.GradientTape() as grad1:
           grad1.watch(x)
           u = self.call(x, t)
      ux = grad1.gradient(u, x)
 uxx = grad2.gradient(ux, x)
 return uxx
```

Example 2: Improved network architecture from [23] + normalization of input data to (-1, 1)

```
class FNN(tf.keras.Model):
    def __init__(self, x_max, t_max, dense_size, num_hid):
       super().__init__()
       self.max = tf.constant([x_max, t_max], dtype=tf.float64)
       self.in0 = tf.keras.layers.Dense(dense_size, activation="tanh")
       self.in1 = tf.keras.layers.Dense(dense_size, activation="tanh")
       self.in2 = tf.keras.layers.Dense(dense_size, activation="tanh")
       self.hid = []
       for i in range(num_hid):
          self.hid.append(tf.keras.layers.Dense(dense_size, activation="tanh"))
       self.out = tf.keras.layers.Dense(1)

    def call(self, x, t):
       inputs = tf.stack([x, t], axis=1)
       in_norm = 2.0 * (inputs / self.max) - 1.0
       h = self.in0(in_norm)
```

```
        u = self.in1(in_norm)
        v = self.in2(in_norm)
        for layer in self.hid:
          z = layer(h)
          h = (1.0 - z) * u + z * v
        out = self.out(h)
        return tf.reshape(out, [-1])
```

Example 3 - one training step for updating network weights $\mathbf{W}$ and model parameters $\boldsymbol{\gamma}$. All gradients are clipped so that their L2 norm does not exceed 1.0

```
@tf.function
def _train_step(self):
      with tf.GradientTape(persistent=True) as grad:
            loss = self.loss_pinn(*self.loss_args)
      wgh_grads = grad.gradient(loss, self.u.trainable_variables)
      pars_grads = grad.gradient(loss, self.trainable_params)

      del grad

      wgh_grads = [tf.clip_by_norm(g, 1.0) for g in wgh_grads]
      pars_grads = [tf.clip_by_norm(g, 1.0) for g in pars_grads]

      # Update the variables
      self.opt_w.apply_gradients(zip(wgh_grads, self.u.trainable_variables))
      self.opt_p.apply_gradients(zip(pars_grads, self.trainable_params))
      return loss
```